SY27-7244-1

Systems

OS/VS2 Supervisor Logic

Volume 1

Release 1.6

IBM

Second Edition (September 1974)

This edition is a minor revision of SY27-7244-0 incorporating TNL SN27-1403. These publications are still current.

This edition applies to release 1.6 of OS/VS2 and to all subsequent releases 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 IBM System/360 and System/370 Bibliography, GA22-6822, and the current SRL Newsletter.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to: IBM Corporation, Programming Publications, Department 636, Neighborhood Road, Kingston, New York 12401

This manual describes the logic of the OS/VS2 Supervisor, its relationship to the other portions of the control program, and the interaction between supervisor modules. The information in this manual is intended for personnel who are responsible for determining sources of error within or making modifications to the VS2 Supervisor.

Two short sections that follow this preface can help you use this manual effectively. "How This Manual Is Organized" describes the organization of the manual: where to find overview material to refresh your memory, and where to find detailed information to correct errors or modify the system. "How to Read the Method-of-Operation Diagrams" explains the format of the diagrams and the symbols that are used in the diagrams.

The communications task and checkpoint/restart logic are not documented in this manual. The communications task is documented in the OS/VS2 Job Management Logic PLM, Order No. SY28-0649; checkpoint/restart logic is documented in the OS/VS Checkpoint/Restart Logic PLM Order No. SY24-5159.

Prerequisite Knowledge and Reading

If you do not know the basic concepts of and services provided by the VS2 Supervisor, you should read the following publication:

OS/VS Supervisor Services and Macro Instructions, GC27-6979

Other Publications to Which the Text Refers

The publications listed below are referred to within the section shown in parentheses. The publications are listed alphabetically according to the first descriptive word in the title.

OS/VS Debugging Guide (Termination), GC28-0632

OS/VS2 Dynamic Support System Logic (Interruption Supervision), SY28-0679

OS/VS2 IPL and NIP Logic (Virtual Storage Supervision), SY27-7243

OS/VS2 Job Management Logic (Termination), SY28-0620

OS/VS OPEN/CLOSE/EOV Logic (Contents Supervision), SY26-3785

OS/VS2 Planning and Use Guide (Introduction), GC28-0600

CS/VS Recovery Management Support Logic (Interruption Supervision), SY27-7252

OS/VS2 Service Aids Logic (Termination), GY28-0635

OS/VS TCAM Logic (Termination), GY30-2039

OS/VS2 TSO Control Program Logic (Timer Supervision and Termination), SY28-0649

Notice to Reader: No IBM publication order numbers are provided in references to publications in the body of this manual. Please use the list above to find the order number for publications you find mentioned in the text.

HOW THIS MANUAL IS ORGANIZED

This manual, which describes the logic of the VS/2 supervisor, is divided into two volumes. Volume 1 (Sections 1 through 9) uses text, conventional illustrations, and method-of-operation diagrams to explain basic concepts and to describe how the supervisor works. In addition, Volume 1 contains a glossary of terms and acronyms used in this publication. Volume 2 contains tables and other reference material that is to be used in conjunction with Volume 1 to isolate supervisor errors.

The sections in Volume 1 are:

1: WHAT THE SUPERVISOR IS AND DOES
Briefly describes the functions performed by the supervisor. This section explains the general operation of the supervisor and describes the relationship of the supervisor to the computer.

The diagrams at the end of this section illustrate how the supervisor processes interruptions and where it passes control to perform the services required by those interruptions. These diagrams are also a visual table of contents for the rest of the diagrams in Volume 1.

2: INTERRUPTION SUPERVISION

3: TASK SUPERVISION

4: CONTENTS SUPERVISION

5: PAGING SUPERVISION

6: VIRTUAL STORAGE SUPERVISION

7: TIMER SUPERVISION

8: TERMINATION
Each of these sections (2 through 8) describes one major area of the supervisor: "Interruption Supervision," "Task Supervision," "Contents Supervision," and so on. Each section explains the basic concepts and general logic of the functional area, including descriptions of operational characteristics such as queuing techniques and the use of control blocks.

The descriptive material in each section is followed by a set of method-of-operation diagrams. The first diagram is a table of contents for the detailed diagrams in the set. The set also includes overview diagrams that illustrate the functional flow and refer you to the more detailed diagrams in the set. These detailed diagrams illustrate the input, processing steps, and output, and they contain symbolic names that tie the processing steps to the program listing. Particularly complex functions are also flowcharted. The flowcharts follow the detailed diagram for that function.

9: GLOSSARY
Defines terms and acronyms used in this publication.

The sections in Volume 2 are:

10: PROGRAM ORGANIZATION

Program Organization Diagrams
Illustrates the program structure for each function. These diagrams contain module names, routine names, and entry points.

Synopses of Routines
Lists each major routine in the supervisor in alphabetic order and provides a short description of the routine.

11: DIRECTORY
Consists of three tables: the "Module Directory," the "Entry Point Directory," and a table of routines invoked by SVC instructions.

12: DATA AREAS
Describes the major data areas and control blocks used by supervisor modules.

13: DIAGNOSTIC AIDS

Registers on Entry and Exit
Lists the register contents upon entry to and exit from each routine. The entries are arranged alphabetically by entry point name.

Control Blocks Referenced and Set Matrix
Cross-references control blocks to the supervisor modules that reference or set them.

Completion Codes, Wait State Codes, and Messages Tables
Lists completion codes, wait state codes, and messages and identifies the function and the segment of code within the function that caused the code or message to be issued.

HOW TO READ THE METHOD-OF-OPERATION DIAGRAMS

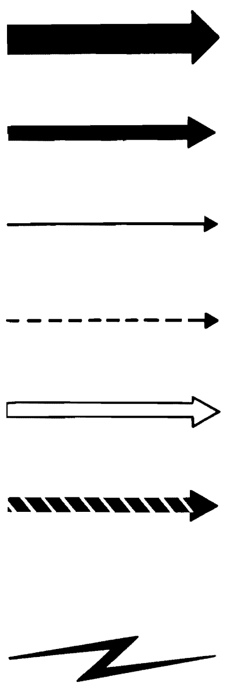
The method-of-operation diagrams illustrate the functions performed by the supervisor. There is a set of these diagrams for each major component. The first diagram in each set is a visual table of contents.

Within each set, the diagrams are arranged in a hierarchy, starting with the least detailed diagram (usually, an overview diagram) and continuing to the most detailed. The detailed diagrams illustrate the input, the processing steps, and the output for each function performed.

The detailed diagrams are read left to right, top to bottom. The input on the left is labeled and boxed, as is the output on the right. The processing is divided into a series of steps, numbered sequentially. If further explanation of a processing step is needed, that explanation appears in the notes on the page facing the diagram. The notes also contain symbolic names so that you can get to the pertinent code in the program listing as quickly as possible.

Arrows are used to signify data movement, data reference, data modification, and processing flow. The arrow conventions are shown in Figure aa.

Other conventions used in the detailed method-of-operation diagrams are illustrated and explained in Figure ab and Figure ac.



Primary Entry/Exit Path -- Shows the path followed to accomplish the principle function of the body of code described by the diagram.

Secondary Path -- Shows the path used by a processing step to accomplish a subordinate function.

Pointer -- Shows that a field in one data area contains the address of another field or data area.

Data Reference -- Shows that the contents of a data area are tested or read to determine the course of subsequent processing.

Data Transfer -- Shows that data from one location is moved or copied into another location.

Control Information Modification -- Shows the setting or changing of switches or pointers that will be used to determine the course of subsequent processing.

Interruption -- Shows the path followed when an I/O, SVC, external/timer, program, or machine interruption occurs.

Figure aa. Legend showing meaning of arrows in method-of-operation diagrams.

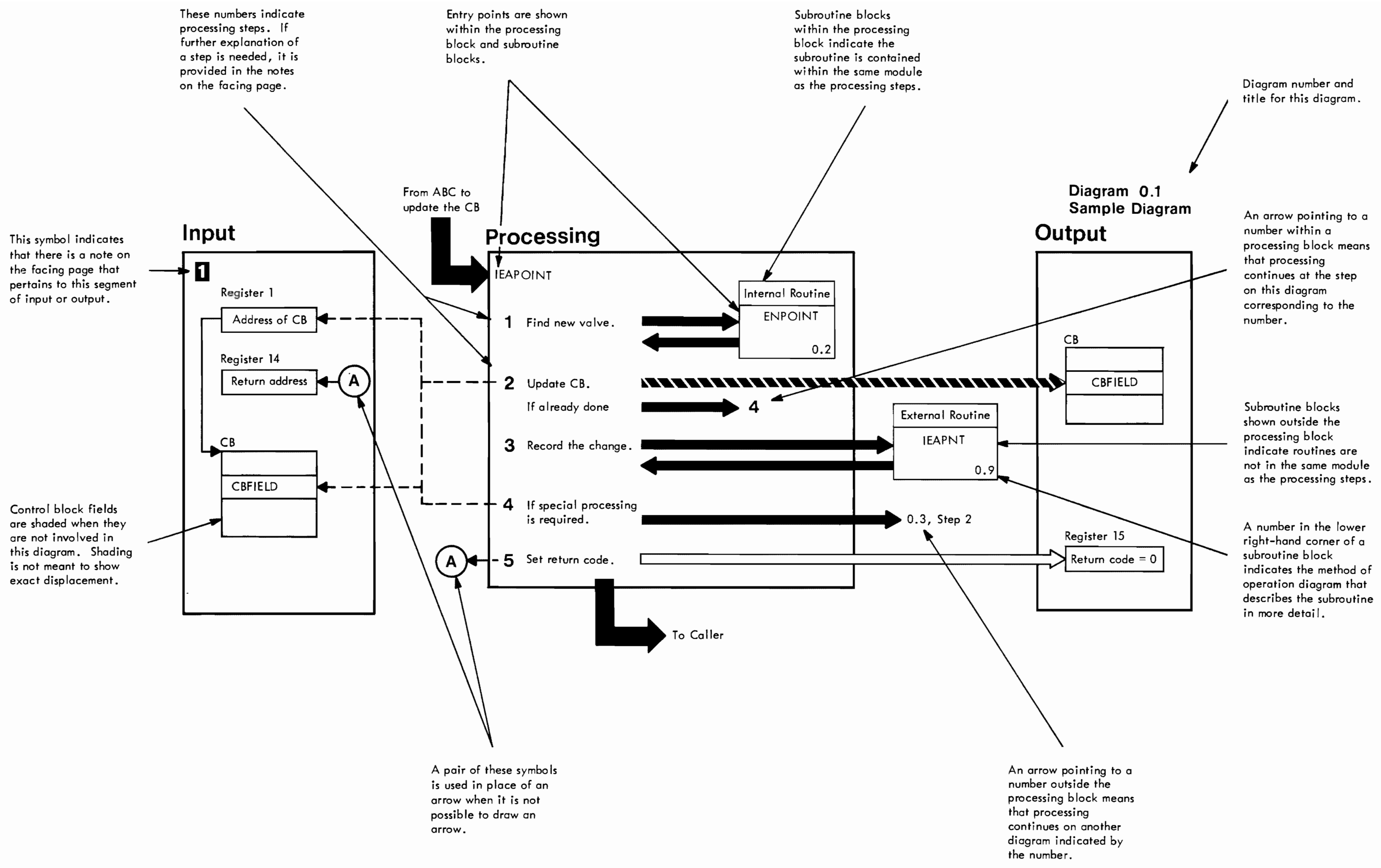
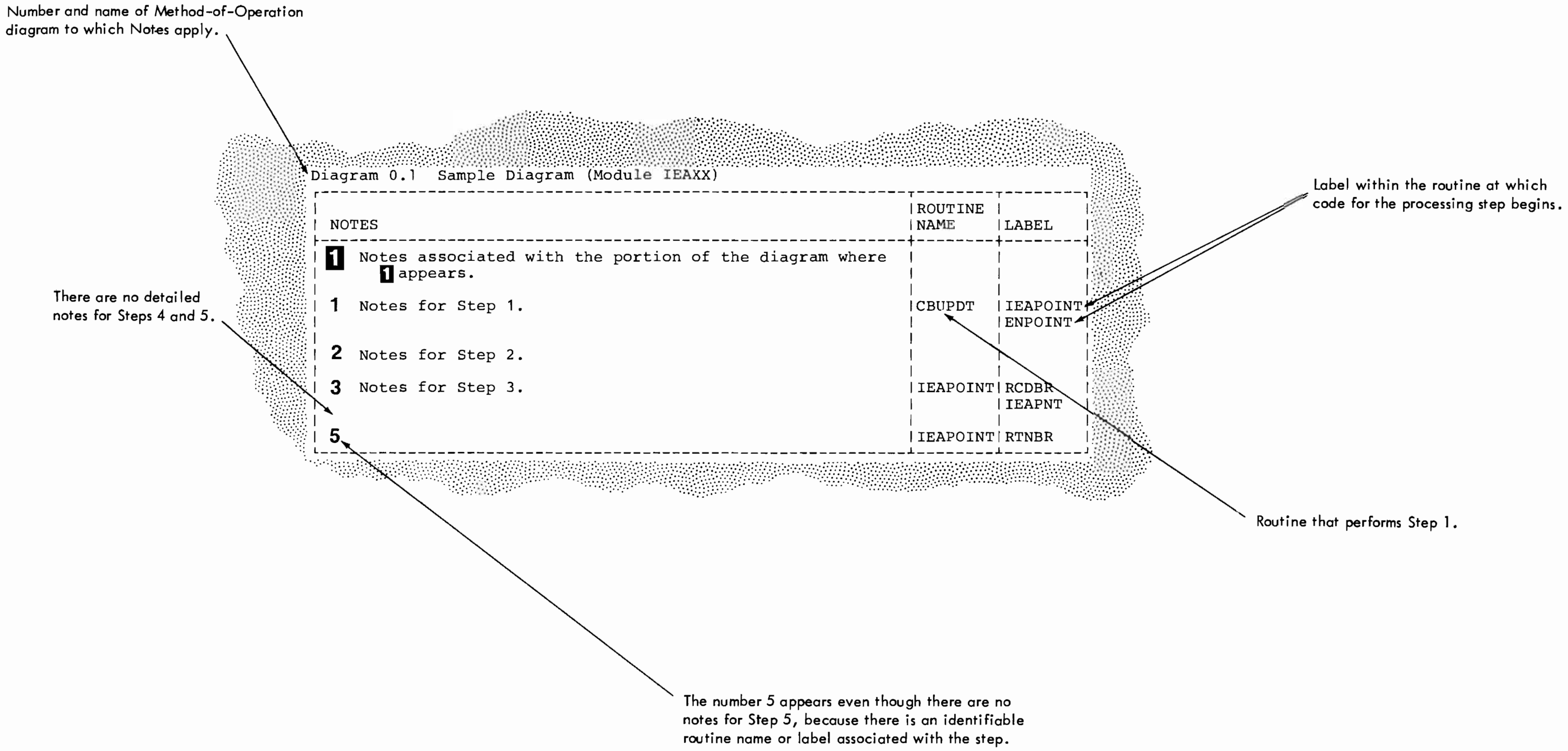

These numbers indicate processing steps. If further explanation of a step is needed, it is provided in the notes on the facing page.
Entry points are shown within the processing block and subroutine blocks.
Subroutine blocks within the processing block indicate the subroutine is contained within the same module as the processing steps.
Diagram number and title for this diagram.
Diagram 0.1
Sample Diagram
This symbol indicates that there is a note on the facing page that pertains to this segment of input or output.
Input
1
Register 1
Address of CB
Register 14
Return address
A
CB
CBFIELD
From ABC to update the CB
Processing
IEAPOINT
1 Find new valve.
2 Update CB.
If already done
4
3 Record the change.
4 If special processing is required.
0.3, Step 2
A
5 Set return code.
To Caller
Internal Routine
ENPOINT
0.2
External Routine
IEAPNT
0.9
Output
CB
CBFIELD
Register 15
Return code = 0
An arrow pointing to a number within a processing block means that processing continues at the step on this diagram corresponding to the number.
Subroutine blocks shown outside the processing block indicate routines are not in the same module as the processing steps.
A number in the lower right-hand corner of a subroutine block indicates the method of operation diagram that describes the subroutine in more detail.
Control block fields are shaded when they are not involved in this diagram. Shading is not meant to show exact displacement.
A pair of these symbols is used in place of an arrow when it is not possible to draw an arrow.
An arrow pointing to a number outside the processing block means that processing continues on another diagram indicated by the number.

Figure ab. Sample Method-of-Operation diagram showing the diagramming conventions used in this manual.



Number and name of Method-of-Operation diagram to which Notes apply.

Diagram 0.1 Sample Diagram (Module IEAXX)

NOTES	ROUTINE NAME	LABEL
1 Notes associated with the portion of the diagram where **1** appears.		
1 Notes for Step 1.	CBUPDT	IEAPOINT ENPOINT
2 Notes for Step 2.		
3 Notes for Step 3.	IEAPOINT	RCDBR IEAPNT
5	IEAPOINT	RTNBR

Label within the routine at which code for the processing step begins.

There are no detailed notes for Steps 4 and 5.

Routine that performs Step 1.

The number 5 appears even though there are no notes for Step 5, because there is an identifiable routine name or label associated with the step.

Figure ac. Sample extended-description page for a Method-of-Operation diagram.

Guide to Section Tabs in Volume 1

ILLUSTRATIONS

CHARTS

SECTION 1

1

Introduction to the Supervisor, and Virtual Storage Concepts

The VS2 supervisor is one part of the control program of the IBM OS/VS2 Operating System; it controls the basic computing system and programming resources needed to perform several data processing tasks concurrently. The supervisor manages real (main) storage and auxiliary storage so that problem programs can occupy and use an address range greater than the address range of real storage. This address space is called virtual storage.

Job steps, designated by the job management routines as tasks, are carried out under the control of the supervisor, which allocates needed resources on the basis of priorities. The supervisor assigns the resources to perform tasks, keeps track of all such assignments, and ensures that the resources are freed upon task completion. If one resource is required by several tasks, the supervisor queues requests for the resource and controls use of the resource. By handling competing demands for resources and by controlling the execution of programs, the supervisor ensures more efficient use of the central processing unit, real storage, auxiliary storage, system and user programs, and other system resources.

Besides the supervisor, the control program consists of the job management, input/output supervision, data management, and recovery management components. The entire control program is introduced in the VS2 Planning and Use Guide.

BASIC CONCEPTS OF VIRTUAL STORAGE SUPERVISION

A virtual storage system introduces a number of new machine and programming concepts. Among these are:

- The concept of virtual storage as opposed to real storage.
- The concept of translating addresses from virtual addresses to real addresses.
- The concept of paging, which is the process of moving fixed-size pages of code and data into and out of real storage.

These concepts are described in the following subsections.

CONCEPT OF VIRTUAL STORAGE

The VS2 supervisor manages virtual storage so that users can effectively use virtual address space as though it were real storage. The virtual storage address space is divided into blocks of 4,096 bytes (4K) called pages. These pages may reside in external page storage (the portion of auxiliary storage used to store pages), in real storage, or both. The pages are transferred into and out of real storage as needed by the system or the users' programs. This process is called paging.

When a program refers to a virtual address, that virtual address is translated by relocation hardware into its corresponding real storage address. If the page containing that virtual address is not in real storage, the CPU generates a page translation exception (a missing page interruption). This interruption notifies the supervisor to bring in the needed page from external page storage. The process of bringing a page into real storage is called a page-in. The entire virtual address translation and paging process are transparent to the problem program, allowing the programmer to use the virtual address space as though it were real storage.

When a page is no longer needed in real storage, it is moved to external page storage to make real storage available for another page. This process is called a page-out. The page is moved to external page storage only if it has been changed; otherwise, it is merely overlaid when a new page is read into storage.

TRANSLATION OF VIRTUAL ADDRESSES

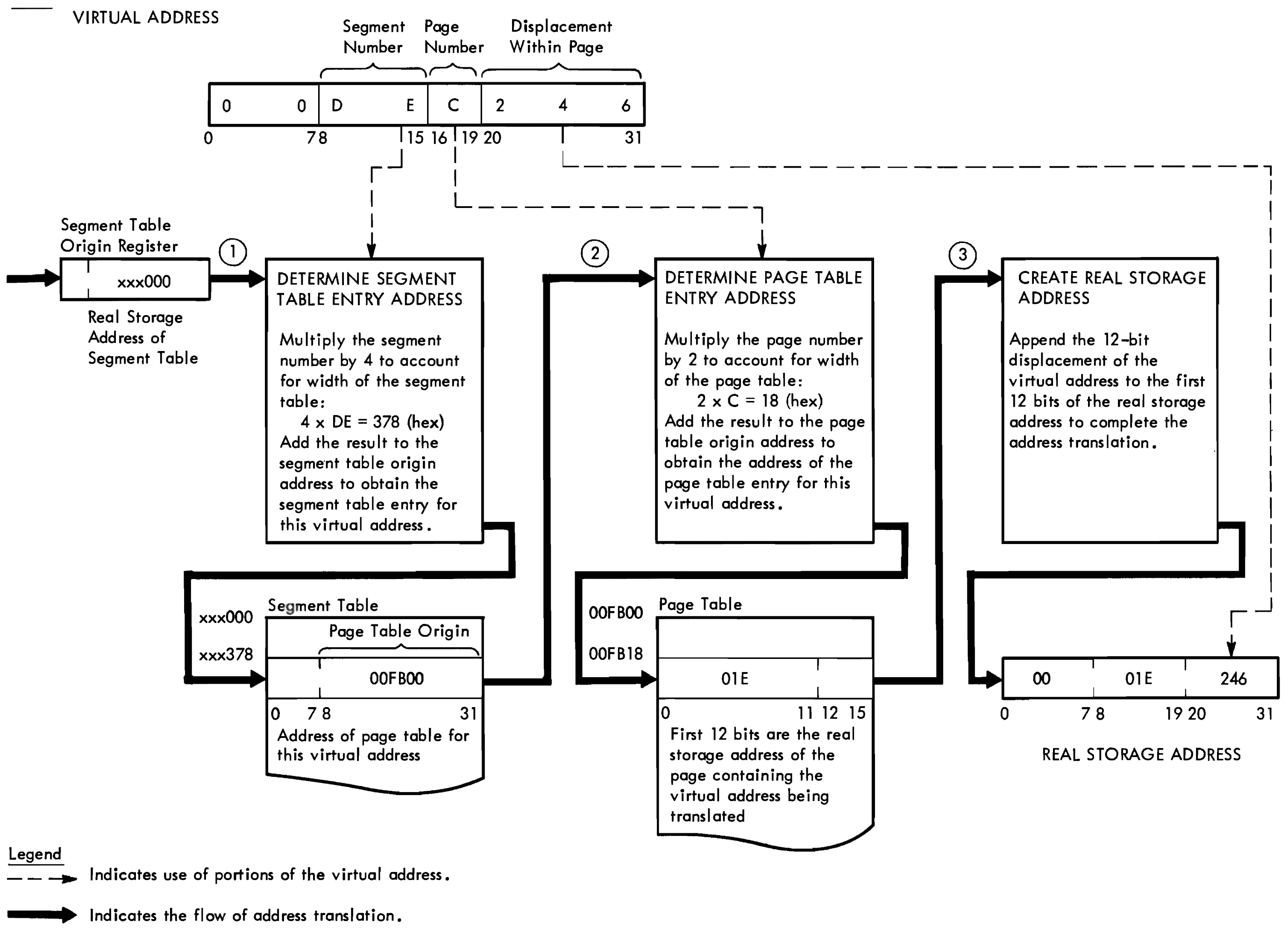
As the CPU executes instructions, the system uses dynamic address translation, a System/370 hardware feature, to translate each virtual address into its real storage equivalent. If the virtual address does not have a real storage equivalent (that is, if the page containing the address is not in real storage), the CPU generates an interruption notifying the supervisor that a page must be brought from external page storage into real storage. Translation occurs only when virtual storage is addressed by the CPU. The entire process is automatic. Storage references by channels for I/O operations are not subject to the translation process.

To facilitate address translation, virtual storage is logically divided in 64K blocks called segments. There are 256 such segments. Each segment is divided into sixteen 4K blocks called pages. This organization enables the dynamic address translation feature to use a two-level table-lookup process to determine the real storage address.

The first-level table used for address translation is the segment table. It contains an entry for each 64K segment of virtual storage. Each segment table entry points to a page table, which is the second-level table used for address translation. Each page table contains 16 entries, one for each 4K page in the segment. The page table entry contains the address of the page in real storage (the first 12 bits of the final translated address). Because each page must reside on a 4K boundary, the last 12 bits of the page address are always zero.

Figure 1-1 shows the translation of a sample virtual address. The virtual address is divided into three parts: the segment number, the page number, and the in-page displacement (number of bytes the address is displaced from the beginning of the page). The dynamic address translation feature translates each virtual address as the CPU executes instructions. To begin the process, the translation feature obtains the origin of the segment table (xxx000 in the example) from a control register called the segment table origin register (step 1 in Figure 1-1). This control register is initialized by the nucleus initialization program, which builds the segment table. The translation feature then uses the segment number (X'DE' in the example) of the virtual address to determine the entry in the segment table for the segment in which that virtual address resides.

The segment table entry (at location xxx378 in the example) points to a page



VIRTUAL ADDRESS
Segment Number
Page Number
Displacement Within Page
0 0 D E C 2 4 6
0 7 8 15 16 19 20 31
Segment Table Origin Register
xxx000
Real Storage Address of Segment Table
1
DETERMINE SEGMENT TABLE ENTRY ADDRESS
Multiply the segment number by 4 to account for width of the segment table:
4 x DE = 378 (hex)
Add the result to the segment table origin address to obtain the segment table entry for this virtual address.
2
DETERMINE PAGE TABLE ENTRY ADDRESS
Multiply the page number by 2 to account for width of the page table:
2 x C = 18 (hex)
Add the result to the page table origin address to obtain the address of the page table entry for this virtual address.
3
CREATE REAL STORAGE ADDRESS
Append the 12-bit displacement of the virtual address to the first 12 bits of the real storage address to complete the address translation.
xxx000
xxx378
Segment Table
Page Table Origin
00FB00
0 7 8 31
Address of page table for this virtual address
00FB00
00FB18
Page Table
01E
0 11 12 15
First 12 bits are the real storage address of the page containing the virtual address being translated
00 01E 246
0 7 8 19 20 31
REAL STORAGE ADDRESS
Legend
Indicates use of portions of the virtual address.
Indicates the flow of address translation.

Figure 1-1. Translation of a virtual address into a real address.

table (at location 00FB00 in the example). The page table contains 16 entries, one for each page in the segment. The translation feature uses the page number (X'C' in the example) of the virtual address to determine which page in the segment contains the virtual address (step 2). This number is used to calculate the address of the page table entry for that page (X'00FB18' in the example). The page table entry contains the real storage address for that page (the first 12 bits of the desired real storage address). As mentioned earlier, a page, when in real storage, must reside on a 4K boundary. Thus, the last 12 bits of the real storage address for a page are zeros.

The in-page displacement portion of the virtual address (X'246' in the example) is appended to the 12-bit address in the page table entry to obtain the desired real storage address (step 3). The CPU then uses the real storage address as it executes instructions.

If the page table entry is flagged invalid, the CPU generates a page translation exception. The page translation exception is a program interruption (code X'11') and indicates that the page containing the virtual address is not in real storage. This interruption (also called a missing page interruption) indicates that the page must be transferred from external page storage to real storage.

If the segment table entry is marked invalid, the CPU generates a segment translation exception. This exception is a program interruption (code X'10') and indicates that: (1) no page table has been created for the segment, or (2) the virtual address being translated is protected from the task referencing it (see "Storage Protection" later in this section).

If, during the translation process, the dynamic address translation feature cannot locate the segment table, the segment table entry, or the page table, the CPU generates a translation specification error. The translation specification error is a program interruption (code X'12'). It indicates that a control program or hardware failure has prevented translation of the virtual address. See Diagram 1.1 to see how missing page page interruptions, segment translation errors, and translation specification errors are processed.

PAGING

In the VS2 system, a program normally executes with only some of its pages in real storage. The specific pages needed at any given time depend upon the structure and addressing pattern of the program. When a page is needed, it is brought into real storage. When no longer needed, it is released or transferred to external page storage. As mentioned earlier, this transfer of pages between real storage and external storage is called paging.

As an instruction is executed, the virtual addresses are translated as explained in the preceding subsection, "Translation of Virtual Addresses". If a virtual address that is being translated resides in a page not in real storage, the CPU generates a missing page interruption. This interruption notifies the paging supervisor, a set of paging routines in the VS2 supervisor, that a page must be brought into real storage (paged in).

When real storage is needed for a page being paged in, the real storage is made available by the paging supervisor. The paging supervisor selects what it determines to be the least needed page and makes the real storage it occupies available for the page-in. If the selected page in real storage was modified during execution, it is actually written out to external page storage (paged out). (The hardware maintains a bit in the storage key that indicates whether the contents of a page frame have been changed.) If the page was not modified and an exact copy of the page already exists on external page storage, the page is not written out; it is merely overlaid by the incoming page.

The paging supervisor maintains three sets of tables to keep track of all pages. The first two sets of tables, the segment tables and the page tables, are used for dynamic address translation. The paging supervisor uses a third table, the external page table, to keep track of all pages on external page storage. For every page table entry there is a corresponding external page table entry which contains the external page storage address at which that page is recorded on a paging device.

The VS2 supervisor does not maintain an image of virtual storage. The pages of virtual storage are actually in real storage, on external storage, or both. They are not stored in contiguous locations or in any sequential order. They are placed in real storage wherever a page frame is available; they are stored on external page storage wherever a page slot is available. The three sets of tables, which are organized according to page addresses, record the physical location of each page.

The paging supervisor maintains a list of available page frames that are least likely to be referenced. When a page frame is needed so that a page can be brought into real storage, the paging supervisor uses one of these available frames. When the number of available frames falls below a value specified during nucleus initialization, the paging routines select the page frames containing the least-used pages and make them available for use. The paging routines use the page replacement algorithm to select the least-used pages, and the process is based on the settings of the change and reference bits in the storage keys associated with each page frame.

The selection of page frames continues until the predetermined number of available page frame is reached. During the process, page frames that contain unreferenced pages are selected before page frames that contain referenced pages. Page frames that contain unchanged pages are selected before page frames that contain changed pages.

If the page in a selected frame has been changed, a copy of the changed page is actually moved to external page storage before the page frame is made available. If the page in a selected page frame has not been changed, a copy is not moved to external page storage, and no page-out is required.

In this illustration, the page frames occupied by Page 8 and Page 4 have been made available. The page table entries for these pages have been marked invalid. Since Page 8 was not changed, there is no need to move it to external page storage and the external storage address remains the same. Page 4, however, has been changed. Therefore, a copy of it is paged out to an available external page storage slot. The external page table entry is updated for Page 4 and the external storage slot formerly occupied by Page 4 (called "Old Page 4" in this illustration) is now available for use when another page-out is necessary. Note that pages are not necessarily transferred to external page storage sequentially.

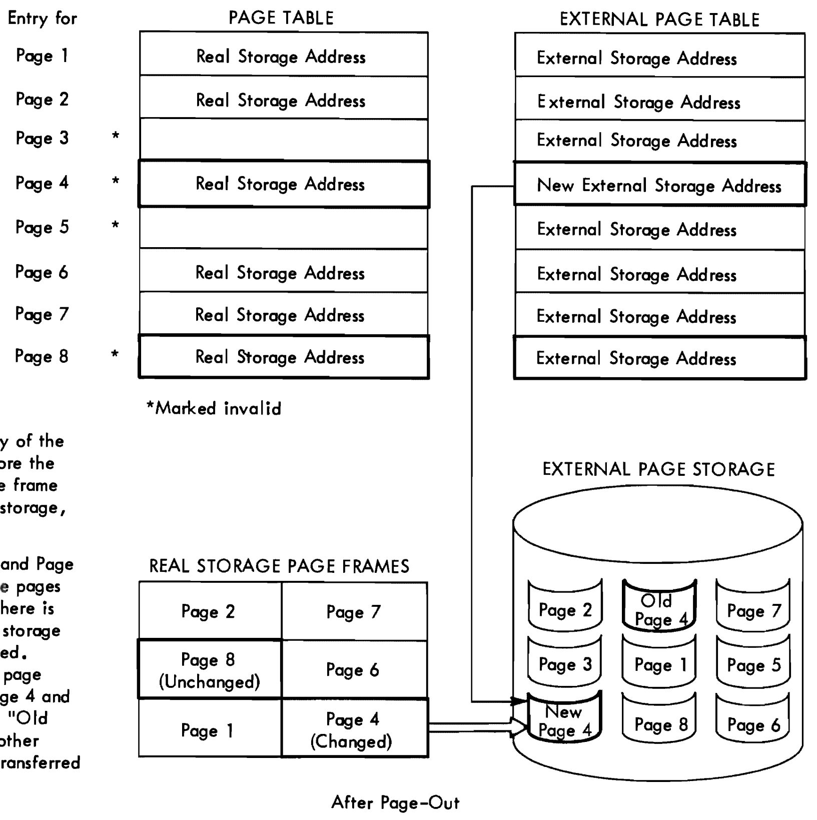


Figure 1-2. Explanation of page-out operations.

Figures 1-2 and 1-3 illustrate the paging process using a virtual storage of eight pages and a real storage of 24K bytes (a real storage large enough for six pages). Real storage is logically divided into 4K blocks called page frames, and each page frame is on a 4K boundary. In the example, real storage consists of six page frames. For purposes of this illustration, the pages are numbered. In the VS2 system, the pages are identified by their page addresses.

ORGANIZATION OF VIRTUAL STORAGE

In its simplest form, virtual storage consists of nothing but address space -- the total 16,277,216 addressable bytes in the system. More concretely, virtual storage consists of pages fixed in real storage, pages stored on the paging devices, and various parameters and control information in the supervisor.

It is the parameters and control information that give form and organization to what otherwise seems to be an amorphous thing called virtual storage.

The system must reserve certain portions of the total address space for specific purposes (for example, space must be reserved for the system nucleus, for the fixed link pack area modules and BLDL entries, and for a minimum system queue area in which to build system control blocks).

The supervisor must also know how much address space can be used for other purposes, such as how much space can be allocated to nonpageable (V=R) tasks and how much to pageable (V=V) tasks.

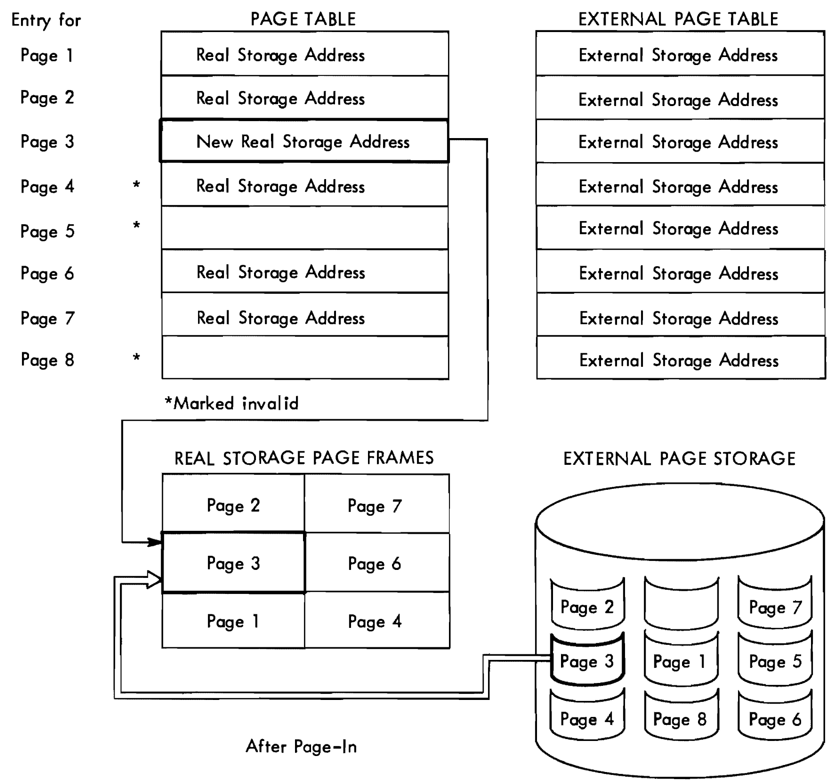
And, while the system is running, the supervisor must keep track of how much storage is being used for the different purposes, and it must ensure that the limits on the different areas are not exceeded.

For these reasons, the supervisor divides the virtual address space into distinct sections. Each section is reserved for a particular purpose, and whenever

When an executing program refers to a virtual address that is not in real storage, the virtual address cannot be translated to a real address, and a missing page interruption is generated. Control is given to the paging supervisor. The paging supervisor first attempts to reclaim the page. If the needed page occupies a page frame that has been made available for a page-in but that page frame has not yet been re-used, the page is reclaimed and a page-in is avoided. The paging supervisor merely updates the page table entry for the reclaimed page, and removes the page frame from the available list (called the available page queue). Page 4, for example, could be reclaimed.

In this illustration, the executing program has referred to a virtual address in Page 3. Since Page 3 was not in real storage (see Figure 1-2), a copy of that page was paged into the page frame formerly occupied by Page 8. The page table entry for Page 3 was updated so that it now contains the real storage address of the page frame now occupied by Page 3. The needed page is now in real storage and when the interrupted program regains control, it can continue executing.

The first time a page is referred to, a page-in is not required. There is no associated page on external page storage. The paging supervisor merely allocates an available page and optionally, clears it.



• Figure 1-3. Explanation of page-in operations.

storage is requested for that purpose, the storage is allocated from the section of virtual address space reserved for that purpose. The size of each section is established during nucleus initialization and is based on parameters received by the system at that time.

The space for one section may exist totally in real storage (as it does for the system nucleus), or it may be a combination of real storage space and pages on paging devices (as it does for a pageable task). Thus, some sections are fixed in real storage and other sections are paged.

Figure 1-4 illustrates the organization of virtual storage. Keep in mind that the illustration depicts the way in which virtual storage looks when all the pages are laid out sequentially. The VS2 supervisor keeps no such image. To construct this image after the system is in operation, you would have to sort through, combine, and order the pages in real storage and external storage using the page tables and other control information as a guide.

The SQA (system queue area) is reserved for non-job-oriented and non-job-step-oriented control blocks and tables (such as the segment tables) that are maintained by the control program. It is reserved in segment multiples. As long as an SQA page is allocated for use, it is nonpageable and is usually fixed in the high end of real storage. The number of SQA pages fixed in real storage increases and decreases to meet the needs of the system.

The pageable LPA (pageable link pack area) contains selected reenterable rourtines that can be used concurrently by all tasks in the system. It is pageable and the fixed link pack area is an extension of this area (see below).

The pageable BLDL table contains the list of directory entries for the SYS1. LINKLIB data set. The pageable BLDL table can exist only if the user does not specify a fixed BLDL table.

The master scheduler region is used by the master scheduler task and the communications task.

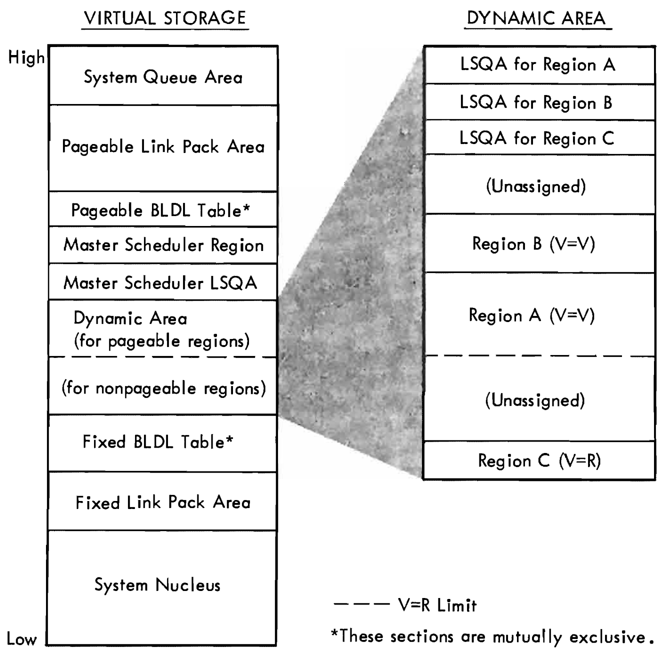


Figure 1-4. Organization of virtual storage.

The master scheduler LSQA is the local system queue area for the master scheduler. (The local system queue area is explained below.)

The dynamic area is used for execution of user and system tasks. The area is divided into two parts; the part below the V=R LIMIT is used for nonpageable regions and the area above the line is used for pageable regions. Regions for nonpageable (V=R) tasks are allocated from the lower part, and regions for pageable (V=V) tasks are allocated from the upper part.

An LSQA (local system queue area) is reserved for each initiator, in the high end of the dynamic area. It is allocated in segment multiples. Each LSQA is used to store job-oriented or job-step-oriented control blocks, including the queues of control blocks that contain the information necessary to keep track of the free and allocated space within the region. The LSQA also contains the page tables and the external page tables for the region. Pages allocated in each LSQA are fixed in real storage and their number increases and decreases to meet the needs of the job or job step.

At system initialization, the user can set the V=R limit by specifying the size of the dynamic area to be reserved for non-pageable (V=R) regions. Problem programs that cannot be paged during execution must occupy V=R regions. These regions are allocated pages below the V=R limit. The virtual addresses of these pages have real storage equivalents. The region is fixed in real storage at these identical real storage addresses, thus ensuring that a missing page interruption never occurs and that a virtual address in that region is identical to the real storage address after translation. Pages for V=V regions are allocated from the dynamic area above the V=R limit.

The fixed BLDL table is a BLDL table that the user has specified to be fixed in real storage. It is nonpageable and the virtual and real storage addresses are identical. A system contains either a fixed BLDL table or a pageable BLDL table, but not both.

The fixed LPA (fixed link pack area) is present only if the operator specifies it at IPL time. It is an extension of the pageable link pack area and is searched first (that is, before the pageable link pack area). It is not pageable and its virtual and real storage addresses are identical.

The system nucleus contains the control program routines that must be permanently fixed in real storage. The nucleus is not paged and begins at virtual address zero. Its virtual and real addresses are identical.

USE OF REAL STORAGE

Figure 1-5 shows the organization of real storage. Page frames in the fixed area contain the system nucleus, the fixed link pack area, and the fixed BLDL table. The real storage addresses of the page frames and the virtual addresses of the pages are identical. The pages are not paged out of real storage, the page frames are not eligible for use by other pages, and copies of the pages are not kept on external page storage.

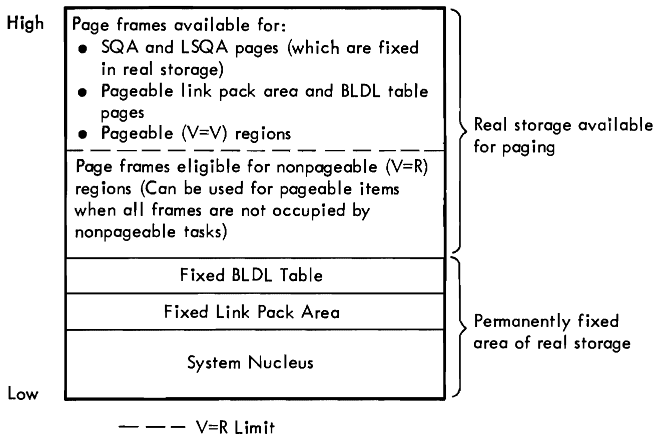

High
Low
Page frames available for:
• SQA and LSQA pages (which are fixed in real storage)
• Pageable link pack area and BLDL table pages
• Pageable (V=V) regions
Page frames eligible for nonpageable (V=R) regions (Can be used for pageable items when all frames are not occupied by nonpageable tasks)
Fixed BLDL Table
Fixed Link Pack Area
System Nucleus
Real storage available for paging
Permanently fixed area of real storage
——— V=R Limit

Figure 1-5. Organization of real storage.

All page frames in the rest of real storage are eligible for paging. However, pages allocated for the SQA (system queue area) or an LSQA (local system queue area) must reside in real storage until they are no longer needed. Thus, page frames containing SQA or LSQA pages are not eligible for paging until those pages are freed. Copies of SQA and LSQA pages are not kept in external page storage (except LSQA pages for TSO regions).

Pages for all pageable programs can use the page frames in the pageable area. These pages are paged out of real storage, the page frames are eligible for use by other pages, and copies of the pages are kept in external page storage. The virtual addresses and the real storage addresses are probably different.

The V=R limit established for virtual storage also applies to real storage. If the user requests a V=R region for a nonpageable task, the pages for that region reside in the page frames below the V=R limit. The pages of the nonpageable program occupy the page frames whose addresses are the same as the virtual addresses of the pages that they contain. The virtual and the real addresses are identical. The pages are not paged out of real storage, the page frames are not eligible for use by other pages until the V=R region is freed, and copies of the pages are not kept on external page storage. When page frames below the V=R limit are not being used for V=R regions, they are eligible for use by other pages. However, if a page that does not belong to a V=R region must be long-fixed in real storage, the supervisor attempts to place that page in a page frame above the V=R limit. In so doing, the supervisor avoids page-frame allocation that inhibits allocation of V=R regions.

STORAGE PROTECTION

Storage protection is a feature that prevents unauthorized access to virtual storage by all except the intended users. The larger addressable storage of the VS2 system allows users to start more than 15 regions, and consequently, storage protection has been extended to cover the additional number of regions. The VS2 system still uses the 16 standard protection keys. It supplements this key protection with a segment validation process.

Key-0 programs have access to all allocated areas of virtual storage. Non-key-0 programs have read-only access to the shared areas of the system (for example, the system nucleus, the SQA segments, and LSQA segments) and full access to their own regions.

System programs (such as the supervisor or data management routines) are assigned protection key 0. Nonpageable programs are assigned protection keys 2 through 15. (Thus the user can run a maximum of 14 nonpageable programs at any one time.) All pageable programs are assigned protection key 1. If a pageable or nonpageable user program attempts to reference fetch-protected or non-key-0 virtual storage assigned to another nonpageable program, the system generates a storage protection interruption. The protection keys prevent nonpageable programs from referencing the non-key-0 areas of all pageable programs and other nonpageable programs. They also protect all pageable programs from all nonpageable programs.

Since all pageable programs are assigned protection key 1, the VS2 system uses a segment validation process to protect these programs from each other. Two segment tables are used: one for all system and nonpageable programs and a second for all non-key-0 pageable programs. Both segment tables point to the same set of page tables. Each segment of virtual storage allocated to a program is represented by a segment table entry in a segment table. All segments allocated to pageable programs are represented by segment table entries in the second segment table.

Whenever a pageable program is dispatched, the dispatcher marks as valid all segment table entries for that program's region. All other segment table entries for nonshared areas are marked invalid. Whenever a pageable program attempts to address virtual storage in a segment whose segment table entry is marked invalid, the system generates a segment translation error during virtual address translation. The system treats a segment translation error as though it were a storage protection check. This process protects pageable programs from each other.

Nonpageable programs cannot be totally isolated. Since key-0 areas are not fetch-protected and since nonpageable programs use the system segment table, nonpageable programs can reference key-0 areas in other regions, including pageable regions.

INTERRUPTION HANDLING

All supervisor activity begins with an interruption. In an IBM System/370, the interruption is a machine characteristic; it is the means by which the supervisor gets control of the CPU to provide resources for the performance of tasks. An interruption may be planned (specifically requested in the program currently being executed by the CPU) or unplanned (caused by an event that may be either related or unrelated to the task currently being performed).

There are five types of interruptions:

- Supervisor call (SVC) interruption: a request for a particular supervisor service.
- Timer/external interruption: an attention signal from the System/370 clock comparator or CPU timer, the console interruption key, or the direct control feature.
- Input/output interruption: the signal that an input/output event has occurrred.
- Program interruption: a signal that a program has attempted an invalid action, has met an exception condition during dynamic address translation, has executed an SSM (set system mask) instruction, or is having interruptions monitored by the dynamic support system (DSS).
- Machine-check interruption: the signal that a machine error has occurred. See the OS/VS Recovery Management Support Logic manual for details.

Overall operation of the supervisor is shown in Figure 1-6.

The program being executed in the performance of task A has been interrupted, possibly because it contained a request for a supervisor service, possibly because an input/output operation has been completed for an entirely different task.

The interruption-handling portion of the supervisor (represented by the top box in Figure 1-6) analyzes the interruption, based on control information passed to it at the time of the interruption. Each of the five interruption types has associated with it two program status words (PSWs) called "old" PSW and "new" PSW. The old PSW contains the information needed by the supervisor to analyze the interruption. The new PSW contains the address of the appropriate interruption-handling routine.

When an interruption occurs, the CPU stores the contents of the current PSW in the old PSW for that type of interruption, and loads the new PSW. By loading the new PSW, the CPU places itself in supervisor state and passes control to the associated interruption-handling portion of the supervisor. The supervisor then passes control to those parts of the control program that perform the services required as a result of the interruption. For a more detailed introduction to interruption handling, see Section 2 and Diagram 1.1.

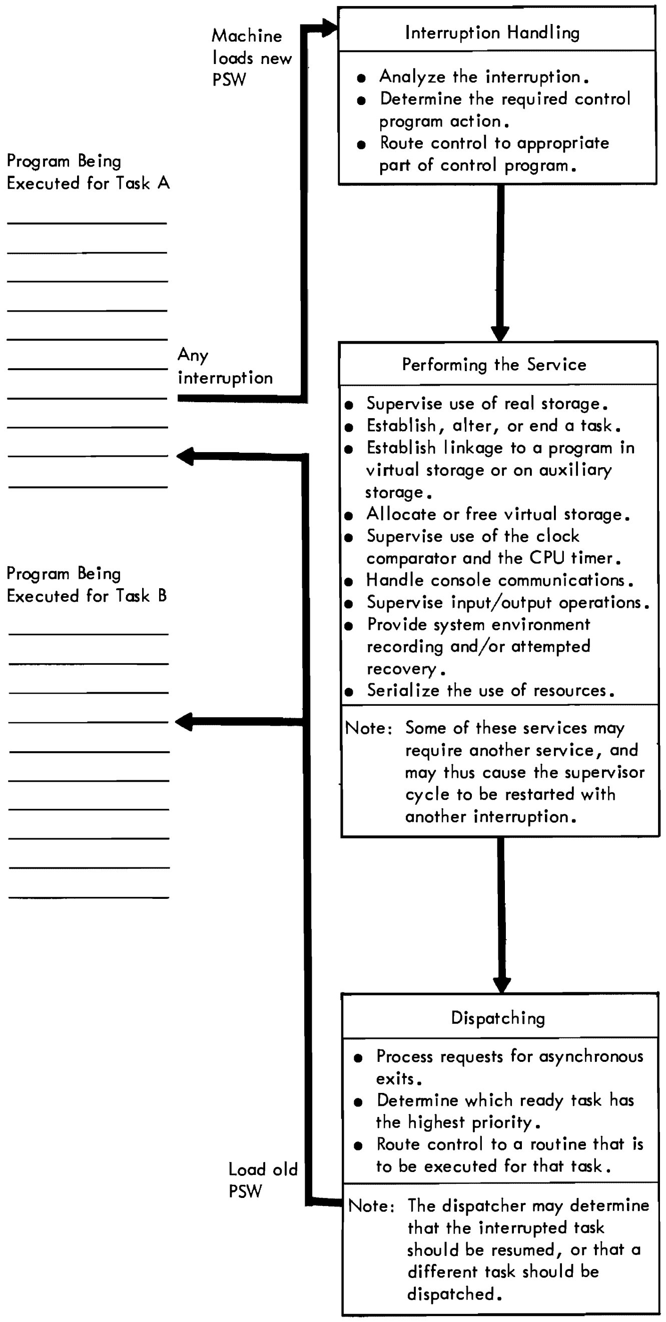

Machine loads new PSW
Interruption Handling
• Analyze the interruption.
• Determine the required control program action.
• Route control to appropriate part of control program.
Program Being Executed for Task A
Any interruption
Performing the Service
• Supervise use of real storage.
• Establish, alter, or end a task.
• Establish linkage to a program in virtual storage or on auxiliary storage.
• Allocate or free virtual storage.
• Supervise use of the clock comparator and the CPU timer.
• Handle console communications.
• Supervise input/output operations.
• Provide system environment recording and/or attempted recovery.
• Serialize the use of resources.
Note: Some of these services may require another service, and may thus cause the supervisor cycle to be restarted with another interruption.
Program Being Executed for Task B
Dispatching
• Process requests for asynchronous exits.
• Determine which ready task has the highest priority.
• Route control to a routine that is to be executed for that task.
Load old PSW
Note: The dispatcher may determine that the interrupted task should be resumed, or that a different task should be dispatched.

Figure 1-6. General flow of the supervisor.

SUPERVISOR SERVICES

The supervisor itself performs many of the services that are requested through an interruption (these services are represented by the middle box in Figure 1-6). Services that the supervisor provides may be grouped into these general categories:

- Task supervision. The task supervision routines create a task in response to a request to attach a subtask to an already existing task. The routines determine in what order tasks are to be executed. Task supervision routines also perform the exiting procedures that prepare for the return of control from a completed program.

- Contents supervision. The contents supervision routines keep records of all programs in virtual storage, and schedule the execution of these programs for tasks. The Program Fetch routine brings requested programs into virtual storage from auxiliary storage.

- Paging supervision. The paging supervision routines manage real storage and external page storage. They ensure that pages are brought into real storage when needed.

- Virtual storage supervision. The virtual storage supervision routines assign virtual storage needed to perform job steps and tasks within job steps.

- Timer supervision. The timer supervision routines control the use of the System/370 clock comparator and CPU timer.

- Task termination. The termination routines perform normal and abnormal termination of tasks.

Most supervisor services are requested by issuing a macro instruction. The last machine instruction of the resulting macro expansion is an SVC instruction. When executed, the SVC instruction causes an SVC interruption. Control passes to the SVC interruption handlers which, in turn, pass control to the routine that performs the requested service. Diagrams 1.1 through 1.3 illustrate SVC processing.

Most SVC routines are executed in a disabled state; that is, they cannot be interrupted by I/O or external events during processing. To avoid missing page interruptions while disabled, these SVC routines have special prologues (preliminary routines) to ensure that all pages needed during their processing are in real storage before disabled processing begins. These SVC prologues use the MODESET service of task supervision to enable interruptions. They then reference all needed virtual addresses. This referencing causes all the pages in which those virtual addresses reside to be paged into real storage by paging supervision. The SVC prologue then issues a MODESET SVC to disable interruptions. The disabled SVC routine can then process the service request without incurring a missing page interruption.

After a control program service has been performed, the supervisor determines which task is to be performed next. The dispatcher (a task supervision routine represented by the bottom block in Figure 1-6) returns control to a processing program or another supervisor routine. As seen in Figure 1-6, the program to which control passes need not be the one that was interrupted. The dispatcher may determine that as a result of the interruption, task B, which has a higher priority than task A, should be performed next.

Each major category of supervisor processing is described more fully in the following subsections.

Task Supervision

Each task to be performed by the system is represented by a TCB (task control block). The TCB contains control and status information related to the task, and pointers to system resources assigned to the task.

When the operating system is generated, six key TCBs are built into the system. These TCBs represent the paging task, the system error task, the dynamic support system task, the communications task, the dynamic device reconfiguration task, and the master scheduler task of job management. All other task control blocks are constructed by the supervisor ATTACH routine, at the request of either the control program or a user program. The master scheduler can attach initiator/terminator tasks. Initiator/terminator routines attach job-step tasks and subtasks. An entire tree structure of related tasks may thus be formed.

All TCBs in the system are chained together, according to dispatching priority, to form the task queue. The paging TCB is at the top of the queue, followed in order by the system error TCB, the dynamic support system TCB, the communications TCB, and the master scheduler TCB. The dispatching priorities of other tasks are assigned by the supervisor according to the parameters given in the ATTACH macro instructions. When several TCBs with the same priority appear in the TCB queue, they are arranged on the queue in descending order in the sequence in which they were created (that is, the first TCB created at that priority is the highest, the second TCB created is the next lower on the queue, and so on).

Figure 1-7 shows the flow of control that results from issuance of the ATTACH macro instruction. This flow is typical of the processing that follows most supervisor macro instructions.

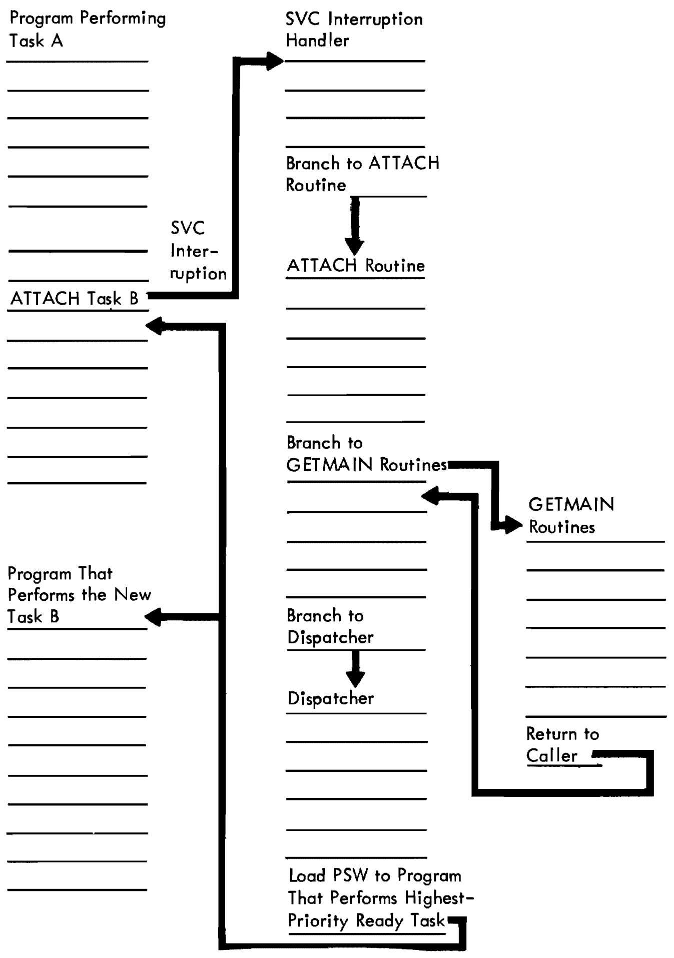


Figure 1-7. Flow of control after the ATTACH macro instruction is issued. (Shows typical processing for a supervisor macro instruction.)

The ATTACH routine, like other SVC routines, is entered as a result of an SVC interruption. The SVC interruption-handling routines analyze the interruption, determine which service is required, and then branch to the ATTACH routine. The ATTACH routine obtains virtual storage for a TCB by branching to the GETMAIN routine. When the required storage has been allocated, the ATTACH routine regains control. It initializes certain fields in the TCB and places it on the task queue.

After the ATTACH routine has initialized the TCB that represents the new task, it branches to the dispatcher. The dispatcher examines the task queue to find the highest-priority task that is ready to be performed. This task may or may not be the one that was being performed at the time the original SVC interruption occurred. For example, if task B has been attached as a subtask of task A, and if B has a higher dispatching priority than A, then task B will be performed before task A is resumed.

The supervisor controls the order in which tasks are performed. This control is accomplished by the dispatcher, working through the task queue. The task queue serves as a record of the status of every task in the system. The highest-priority task represented on the task queue may not be the one to be dispatched; it may be waiting for some event (through the WAIT macro instruction, for example) or for a resource that has been serialized (through the ENQ macro instruction). To be dispatched, a task must be the highest-priority ready task on the queue (that is, the highest-priority task that is currently ready to begin processing).

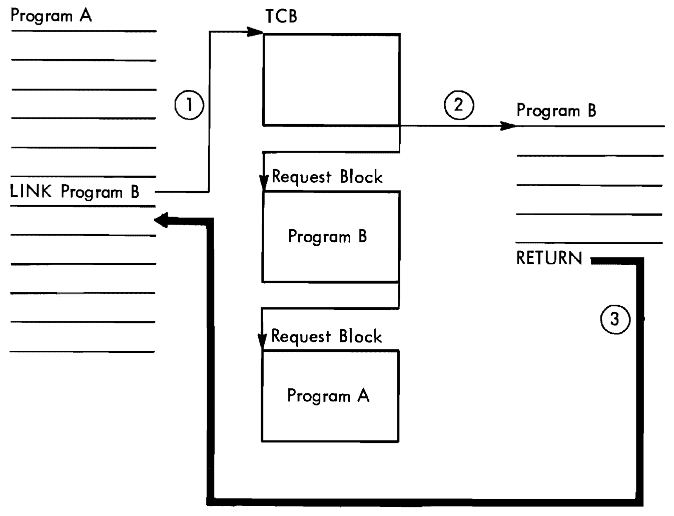
The time-slicing feature is included in the system, and the dispatcher contains code to perform time-slicing. The dispatcher controls time-slicing through the TSCE (time-slice control element); there is one TSCE, assembled at system generation, for each time-slice group.

The user can also identify a single priority level as an APG (automatic priority group) at system generation or system initialization. However, a time-slicing group cannot be identified as an automatic priority group. The dispatcher dispatches tasks within the automatic priority group according to a special algorithm that attempts to provide optimum use of CPU and I/O resources by these tasks. For more information on automatic priority groups, see "Automatic Priority Grouping" under "Special Features" in this section.

Task supervision also provides routines that prepare for the return of control from a completed program and perform the actual return of control. Control may return to a mainline program or to a supervisor routine. The exiting procedures determine the type of program that has completed its execution, and perform different clean-up operations according to the type of program.

The dispatcher is usually entered to return control to a program belonging to the highest-priority ready task. There are some cases, however, in which the dispatcher is not entered to return control: for instance, when the completed program is a type-1 SVC routine that has not indicated the need for a task switch.

For a more detailed introduction to task supervision, see Section 3.



(1) Program A issues a LINK macro instruction specifying Program B.

(2) Contents supervision routines construct a new RB to represent Program B's level of control.

(3) When Program B completes processing, control passes back to Program A.

Figure 1-8. Use of the RB queue in passing control to different programs.

Contents Supervision

Contents supervision is accomplished through a structure of queues that are very closely related to the task queue. These are the request block queues.

Request blocks (RBs) represent levels of control within a task. Contents supervision routines construct an RB for the first level of control in a new task (the first program to be executed for a task): this RB and RBs for subsequent levels are chained on the TCB's RB queue. The subsequent-level programs are programs that are called by higher-level programs and must be executed before the higher-level programs can be completed. Figure 1-8 illustrates the process. If program A issues a LINK macro instruction specifying program B, contents supervision routines construct a new RB to represent program B's level of control. The supervisor uses the RB queue as a record of control levels. As new programs are called, the supervisor can pass control to the succeeding levels and, as the routines complete their operation, it can pass control back up the line, regardless of the number of times a task is interrupted.

There are five types of RBs:

- IRBs (interruption request blocks), which control routines that must be executed in the event of asynchronous interruptions or events. IRBs are created in advance of an interruption by the CIRB routine at the user's request, but are not placed on an RB queue until an interruption or event actually occurs.

- PRBs (program request blocks), which represent nonsupervisory routines that must be executed in the performance of a task. PRBs are created by the contents supervision routines that perform the ATTACH, LINK, SYNCH, and XCTL functions.

- SVRBs (supervisor request blocks), which represent supervisor routines. SVRBs are created by the SVC interruption-handling routines. They are queued in the same way as PRBs.

- SIRB (system interruption request block), which is used only for the system error task. There is only one SIRB in the system.

- TIRBs (task interruption request blocks), which represent control program services that must be performed asynchronously. The ATTACH routine creates one TIRB for each task.

Contents supervision routines construct only one type of RB: the PRB.

The supervisor maintains a record of all programs in virtual storage -- their attributes, locations, and use statuses. This record is called the contents directory. The contents directory is made up of four items: (1) the LPD (link pack area directory); (2) the LPAQ (link pack area queue); (3) the JPAQ (job pack area queue); and (4) the load list. Figure 1-9 illustrates the queuing structure of the contents directory.

The LPD is a record of every program in the pageable link pack area. The link pack area contains reenterable routines specified by the control program or by the user. It is loaded by the nucleus initialization program from the SYS1.LPALIB data set. The programs in the link pack area can be used repeatedly by any task in the system.

The entries in the LPD are LPDEs (link pack directory entries). The LPD resides in the link pack area and is pageable.

The LPAQ is a record of every program in the link pack area that is presently in use. The entries in the LPAQ are CDEs (contents directory entries). When a program represented in the LPAQ is requested for a task, it will normally be represented in the task's RB queue by a PRB; the address of this PRB will be inserted in the CDE. The LPAQ resides in the system queue

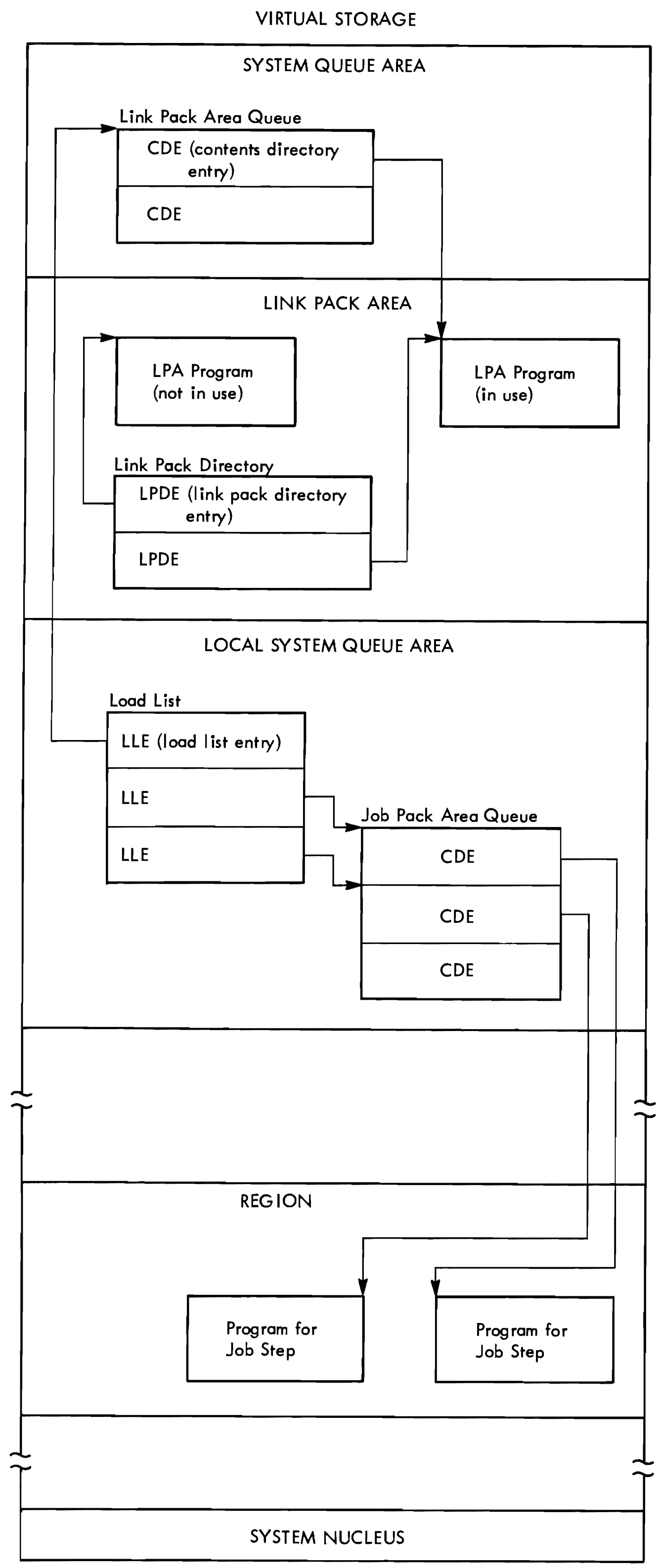


Figure 1-9. The contents directory.

area and is not paged. Whenever a link pack program is requested for use, the LPDE for that program is used to create a CDE which is placed on the LPAQ. The CDE is deleted from the LPAQ when the program is no longer needed. The LPAQ reduces the amount of paging necessary to locate a link pack program.

The load lists represent programs that tasks in regions have requested by issuing LOAD macro instructions. Each requested program was either found in the link pack area or was loaded into the job pack area in virtual storage. The entries in the load list are LLEs (load list elements), not CDEs. Each load list element is associated with a CDE in the JPAQ or LPAQ; the programs represented in the load list are thus also represented in one of the other queues.

There is a JPAQ for each job step in which the LOAD macro instruction has been used to load programs into the job pack area. The JPAQ, like the LPAQ, is made up of CDEs. The JPAQ is used to record the locations of and control the use of routines in the job pack area. These routines can be either reenterable or not reenterable, and they are normally used more than once by routines in the job step. The JPAQ resides in the local system queue area for the job-step region and is not paged.

For a more detailed introduction to contents supervision, see Section 4.

Paging Supervision

Paging supervision routines manage the exchange of pages between real storage and external page storage (see "Paging" under "Basic Concepts of Virtual Storage" earlier in this section). The entire paging process is transparent to the problem program.

To make the most efficient use of real storage and minimize the amount of CPU time used for paging, the paging supervision routines use two algorithms: the page replacement algorithm and the task disablement algorithm. The page replacement algorithm is used to find and release least-used pages in real storage when page frames must be made available for other, more needed, pages. The task disablement algorithm is used to determine whether excessive paging is being done. If the rate of paging is excessive, one of the lower-priority active tasks is set nondispatchable to lessen the contention for real storage. When the paging rate has returned to a more normal level, the tasks are set dispatchable again.

If a missing page interruption can be satisfied by reclaiming the page in real storage, it is. However, if further processing is required before the needed page can be paged in, paging supervision routines construct a PCB (page control block). This PCB represents the paging request and records the status of the needed page. It is queued to one of ten PCB queues. Eight of the queues have associated paging processors, each of which handles the particular paging function required by the PCBs on that queue. When the paging routines have completed processing, the page is in real storage and the user continues processing.

For a more detailed introduction to paging supervision, see Section 5.

Virtual Storage Supervision

The supervisor controls tasks through the task queue and the RB queues; it controls programs through the RB queues and the contents directory. Another major function, controlling virtual storage, is accomplished through a series of virtual storage queues.

When the job management routines designate a job step as a task, they request a region of virtual storage to be used in performing that task. The size of the region is specified by the user; the region contains the job pack area for the step, and all additional working space needed.

All requests for virtual storage are handled by the GETMAIN routines. The supervisor maintains virtual storage queues to reflect storage assignments; the GETMAIN routines simply adjust these queues to reflect new assignments.

When there are no job steps in the system, all of the dynamic area of virtual storage is treated as one region. It is represented to the supervisor by an FBQE (free block queue element) and a PQE (partition queue element) in the system queue area. The PQE contains the address of the FBQE, and the FBQE contains the address of the beginning of the free area. When space is requested for a job step, the GETMAIN routines subtract the requested area size from the free area and build a new FBQE and PQE for the new region. The address of the PQE is placed in the TCB of the job-step task.

After job-step initialization, a program performing a task may request virtual storage by issuing the GETMAIN macro instruction. The GETMAIN SVC routine allocates the storage only within the region assigned to the job step or within the local system queue area or the system queue area. Storage is allocated in the local system queue area and the system queue area only if the requester is a supervisor routine.

The supervisor maintains a separate chain of queue elements for allocation within a region. This chain keeps track of subpools within the region. A subpool is all of the virtual storage requested under a label called a subpool number. All of storage in a subpool does not need to be requested at the same time nor does it need to be contiguous. The chief advantages of subpools are: (1) that the storage can be shared between tasks, and (2) that all of the storage identified by a subpool number can be released with one FREEMAIN macro instruction.

The supervisor FREEMAIN routines are used to free virtual storage when it is no longer needed. Space assigned to a job step, space within a region, space within the local system queue area, and space within the system queue area are all freed by the FREEMAIN routines. These routines make space available by adding elements to chains in which are recorded all free areas in virtual storage, and by adjusting the queues of allocated space.

For a more detailed introduction to virtual storage supervison, see Section 6.

Timer Supervision

Timer supervision routines support the System/370 time-of-day clock, clock comparator, and CPU timer features. These routines enable the programmer to obtain the date and time of day, measure periods of time, or schedule activity for a specific time of day. The routines are executed as a result of the macro instructions TIME, STIMER, and TIMER, and are handled just like any other SVC routines.

Timer supervision maintains two queues of TQEs (timer queue elements): one for task-timing requests and the other for real- and wait-timing requests. The TQEs are constructed by the STIMER routine, and each element represents a request for a type of timed interval. The TQE is placed on the appropriate queue in the order in which the requested interval expires. When a time interval expires or a particular time of day is reached, a timer interruption occurs. This interruption causes the Timer Interruption Handler to remove the top element from the appropriate timer queue and to determine what action to be taken. Examples of required action are: (1) scheduling a timer exit, or (2) making a task ready and eligible for dispatching.

For a more detailed introduction to timer supervision, see Section 7.

Task Termination

The supervisor performs the processing needed when a task is terminated, either normally or abnormally. The termination processing includes releasing system resources that were assigned to the task.

The End of Task (EOT) routine performs normal termination processing. Abnormal termination is performed by the ABTERM and ABEND routines. The STA Services and ASIR (ABEND/STA Interface) routines provide a means for recovery from errors causing abnormal termination. The ABDUMP and SVCDUMP routines provide a dump of control blocks and virtual storage related to the terminating task.

For a more detailed introduction to termination, see Section 8.

SPECIAL FEATURES

The following subsections describe special program features provided in the supervisor.

Authorized Program Facility

The APF (authorized program facility) restricts the use of sensitive system service to authorized users. The authorization consists of a code designated by the installation, applied at the job-step level, and used with programs residing in the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the link pack area. The facility is supported primarily by contents supervision and task supervision.

Both problem programs and system programs can test whether a task has authorization to use a specific function by issuing the TESTAUTH SVC macro instruction (SVC 119).

Automatic Priority Grouping

The automatic priority grouping mechanism operates within the dispatcher. The user may specify a single dispatching priority as an APG (automatic priority group). Tasks within the APG are dispatched in a way that makes best use of the system's CPU and I/O capabilities. The dispatcher identifies these tasks as I/O-oriented or CPU-oriented, according to which facility they use more (paging I/O is excluded from this determination). The dispatcher shifts the APG tasks on the task queue so that the I/O-oriented tasks are dispatched first. For more information, see "Section 3: Task Supervision."

Shared Direct Access Storage Device

The shared DASD (shared direct access storage device) feature enables independently operating data processing systems to share direct access storage devices. The feature employs the two-channel switch and its control commands, device reserve and device release. The shared DASD feature is a general name for control program functions that actually reserve and release each device. Essentially, this feature controls the use of a serially reusable resource, the device and the shared data.

System Management Facilities

SMF (system management facilities) is a standard feature of the control program. It gathers and records information used to evaluate system, data set, and direct access volume usage. SMF functions are performed by job management, input/output support, direct access device storage management, and supervisor routines. The supervisor performs the following SMF functions:

- Maintains a record of system wait time.
- Assists in handling time limit expirations.
- Counts and records references to user data sets.
- Controls the output limit for SYSOUT data sets.
- Maintains a record of virtual and real storage usage.
- Records paging statistics.

Time Sharing Option

TSO (the time sharing option), a standard OS/VS2 facility, adds general-purpose time sharing to the facilities available with the VS2 control program, and the supervisor supports TSO operations. This facility enables users at remote terminals to execute programs concurrently and to interact with those programs during execution. The installation can dedicate its system to time-sharing operations or it can run time-sharing and batch-processing operations concurrently.

The time sharing option consists of a control program (containing IBM-supplied service routines and command processors) and a number of IBM Program Products (available from IBM for a license fee). The TSO control program, an extension of the VS2 control program, does the following:

- Identifies and verifies the time sharing user.

- Defines the user job.

- Assigns the user job an amount of execution time (as determined by installation, and called the selected scheduling algorithm).

- Ensures a specified percentage of execution time for batch-processing operations (when required).

- Logically disconnects the user job from the operating system before the job is swapped out of real storage (called "quiescing" the job); logically reconnects the user job to the operating system when the job is swapped into real storage (called "restoring" the job.)

- Establishes and maintains communications between the terminal user, his programs, and the TSO control program.

- Handles attention interruptions from the terminal user.

- Provides an optional set of SMF records for the TSO system.

- Allows optional system control task exits to installation-supplied routines.

- Provides an optional record of the information that is passed to the Time Sharing Driver via the Time Sharing Interface Program from such TSO system routines as the Region Control Task, the Time Sharing Control Task, and the Time Sharing Dispatcher.

For a complete description of the TSO control program, see the _OS/VS2 TSO Control Program Logic_ manual.

Time Slicing

The time-slicing mechanism operates within the dispatcher. A priority is assigned to a group of tasks which are to be time-sliced; time-slicing occurs only among the tasks in the group and only when the priority level of the group is the highest priority level that has a ready task. Each task in the group is dispatched for the specified time slice. The dispatching of tasks within the group continues until all the tasks are waiting, or a task of higher priority than that of the group becomes ready.

The group of tasks to be time-sliced, the length of the time slice, and the priority of the time-sliced tasks are specified by the installation. Any task in the system that is not defined within the time-sliced group is dispatched under the regular conditions; that is, the task is dispatched when it is the highest-priority ready task, and it remains active until it either waits or until a task of higher priority becomes ready.

Tracing Facilities

Two facilities, the trace table facility and GFT (generalized trace facility), are provided to assist in tracing program flow by monitoring and recording system events.

Trace Table Facility: The trace table facility is an optional feature specified at system generation or nucleus initialization. The trace table routines place entries, each of which is associated with a certain type of event, in a trace table. The size of the table is also a system generation or NIP option; when the table is filled, the routine overlays old entries with new entries beginning with the oldest entry in the table. Trace table entries are formatted and printed out on SNAP dumps and stand-alone dumps. For more information, see "Section 2: Interruption Supervision."

Generalized Trace Facility: The generalized trace facility is invoked as a system task when the operator issues the START command. When GTF is started, the operator selects specific events to be traced and may select to record the trace data either in virtual storage or on an external device. When the internal storage option is selected, the recorded data is comparable to that provided by the trace table facility. When the external device storage option is selected, the recorded data is more comprehensive.

When GTF is active, the trace table facility, if present, is disabled. When the trace records are maintained in virtual storage, ABDUMP provides formatted trace data for abnormally terminated programs when a SYSABEND DD statement has been included. When trace records are stored on an external device, a trace EDIT function of IMDPRDMP can be used to provide the output of selected data.

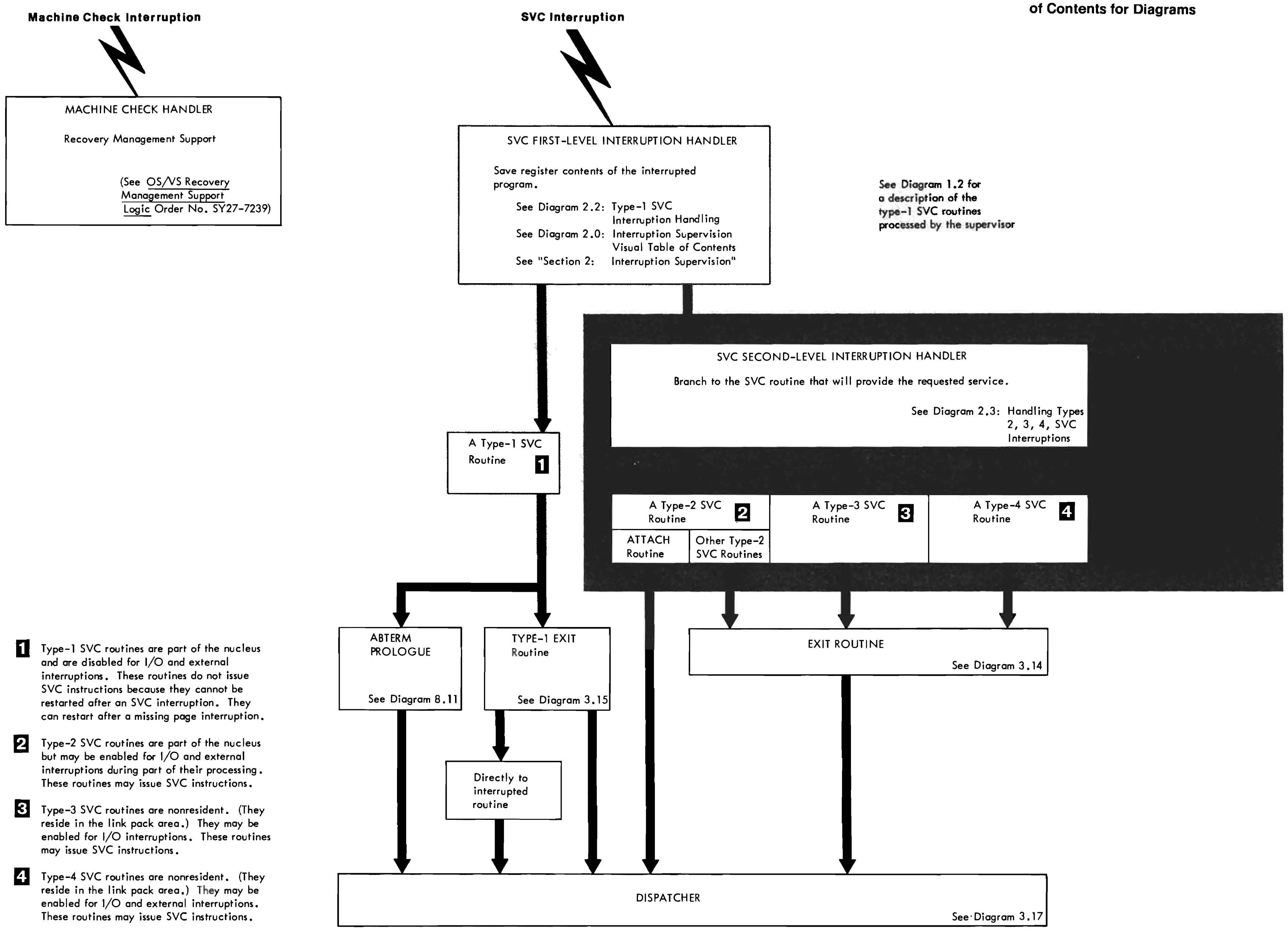
Support for GTF is included in the interruption supervision function of the supervisor (see Section 2.) For a complete description of GTF logic, see the _OS/VS Service Aids Logic_ manual.

SUMMARY OF THE SUPERVISOR'S ROLE

The supervisor is a collection of programs for handling interruptions and providing services in response to them. These interruptions are the basic method by which the control program manages data processing tasks. The supervisor functions are largely performed by routines that manipulate a network of control queues -- the task queue, the RB queues, the contents directory, the paging queues, the real storage queues, the virtual storage queues, the ENQ/DEQ resource queues, and the timer queues.

The processing after a timer/external, input/output, program, or machine-check interruption is generally straightforward. The processing after an SVC interruption is more complicated. Diagrams 1.1 through 1.3 illustrate supervisor interruption processing.

Diagram 1.1 (Page 1 of 3)
Supervisor Overview and Visual Table of Contents for Diagrams


Machine Check Interruption
MACHINE CHECK HANDLER
Recovery Management Support
(See OS/VS Recovery Management Support Logic Order No. SY27-7239)
SVC Interruption
SVC FIRST-LEVEL INTERRUPTION HANDLER
Save register contents of the interrupted program.
See Diagram 2.2: Type-1 SVC Interruption Handling
See Diagram 2.0: Interruption Supervision Visual Table of Contents
See "Section 2: Interruption Supervision"
See Diagram 1.2 for a description of the type-1 SVC routines processed by the supervisor
SVC SECOND-LEVEL INTERRUPTION HANDLER
Branch to the SVC routine that will provide the requested service.
See Diagram 2.3: Handling Types 2, 3, 4, SVC Interruptions
A Type-1 SVC Routine 1
A Type-2 SVC Routine 2
ATTACH Routine
Other Type-2 SVC Routines
A Type-3 SVC Routine 3
A Type-4 SVC Routine 4
ABTERM PROLOGUE
See Diagram 8.11
TYPE-1 EXIT Routine
See Diagram 3.15
EXIT ROUTINE
See Diagram 3.14
Directly to interrupted routine
DISPATCHER
See Diagram 3.17

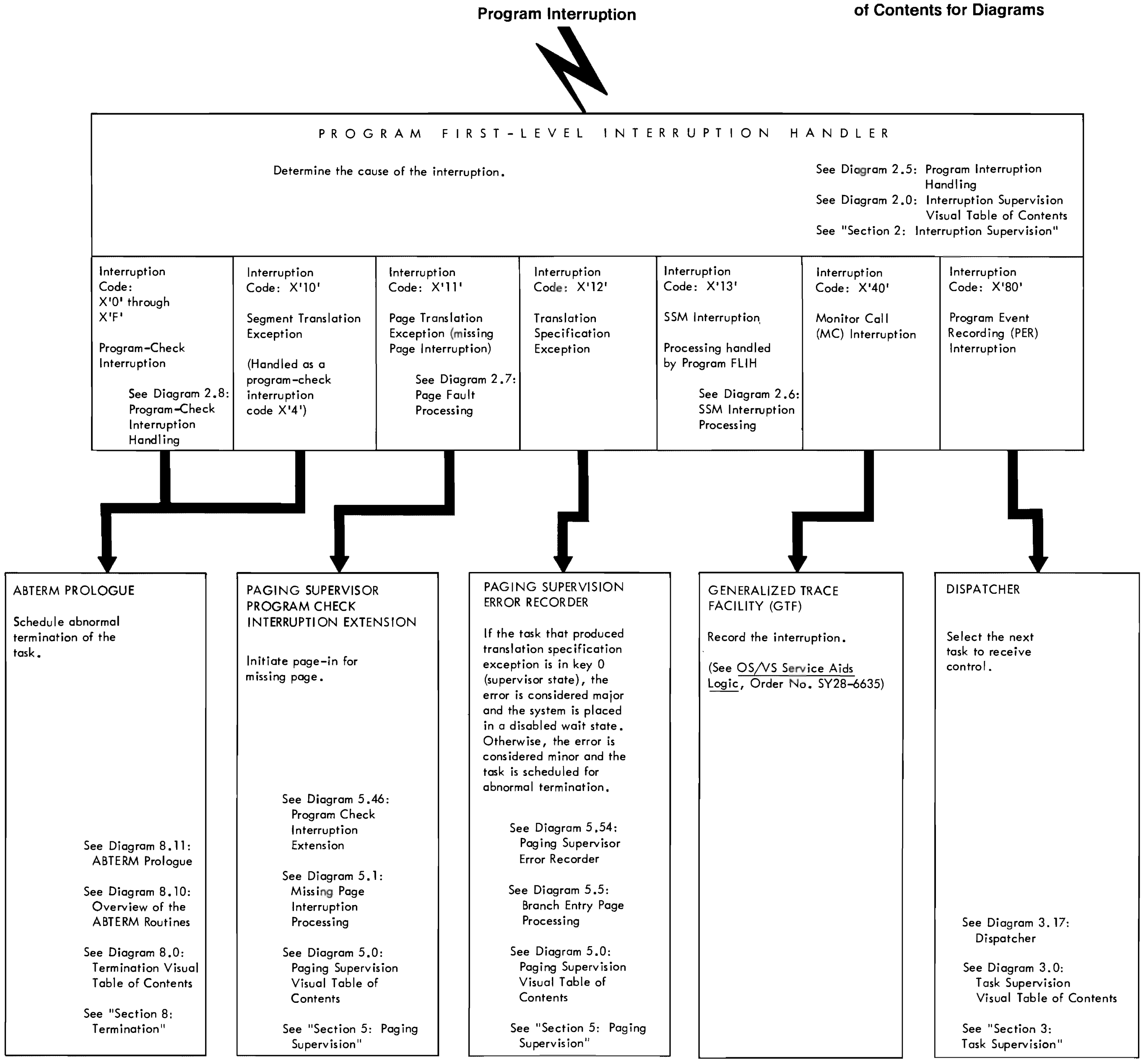
1 Type-1 SVC routines are part of the nucleus and are disabled for I/O and external interruptions. These routines do not issue SVC instructions because they cannot be restarted after an SVC interruption. They can restart after a missing page interruption.

2 Type-2 SVC routines are part of the nucleus but may be enabled for I/O and external interruptions during part of their processing. These routines may issue SVC instructions.

3 Type-3 SVC routines are nonresident. (They reside in the link pack area.) They may be enabled for I/O interruptions. These routines may issue SVC instructions.

4 Type-4 SVC routines are nonresident. (They reside in the link pack area.) They may be enabled for I/O and external interruptions. These routines may issue SVC instructions.

Diagram 1.1 (Page 2 of 3)
Supervisor Overview and Visual Table of Contents for Diagrams



Program Interruption

PROGRAM FIRST-LEVEL INTERRUPTION HANDLER

Determine the cause of the interruption.

See Diagram 2.5: Program Interruption Handling
See Diagram 2.0: Interruption Supervision Visual Table of Contents
See "Section 2: Interruption Supervision"

Interruption Code: X'0' through X'F'

Program-Check Interruption

See Diagram 2.8: Program-Check Interruption Handling

Interruption Code: X'10'

Segment Translation Exception

(Handled as a program-check interruption code X'4')

Interruption Code: X'11'

Page Translation Exception (missing Page Interruption)

See Diagram 2.7: Page Fault Processing

Interruption Code: X'12'

Translation Specification Exception

Interruption Code: X'13'

SSM Interruption

Processing handled by Program FLIH

See Diagram 2.6: SSM Interruption Processing

Interruption Code: X'40'

Monitor Call (MC) Interruption

Interruption Code: X'80'

Program Event Recording (PER) Interruption

ABTERM PROLOGUE

Schedule abnormal termination of the task.

See Diagram 8.11: ABTERM Prologue

See Diagram 8.10: Overview of the ABTERM Routines

See Diagram 8.0: Termination Visual Table of Contents

See "Section 8: Termination"

PAGING SUPERVISOR PROGRAM CHECK INTERRUPTION EXTENSION

Initiate page-in for missing page.

See Diagram 5.46: Program Check Interruption Extension

See Diagram 5.1: Missing Page Interruption Processing

See Diagram 5.0: Paging Supervision Visual Table of Contents

See "Section 5: Paging Supervision"

PAGING SUPERVISION ERROR RECORDER

If the task that produced translation specification exception is in key 0 (supervisor state), the error is considered major and the system is placed in a disabled wait state. Otherwise, the error is considered minor and the task is scheduled for abnormal termination.

See Diagram 5.54: Paging Supervisor Error Recorder

See Diagram 5.5: Branch Entry Page Processing

See Diagram 5.0: Paging Supervision Visual Table of Contents

See "Section 5: Paging Supervision"

GENERALIZED TRACE FACILITY (GTF)

Record the interruption.

(See _OS/VS Service Aids Logic_, Order No. SY28-6635)

DISPATCHER

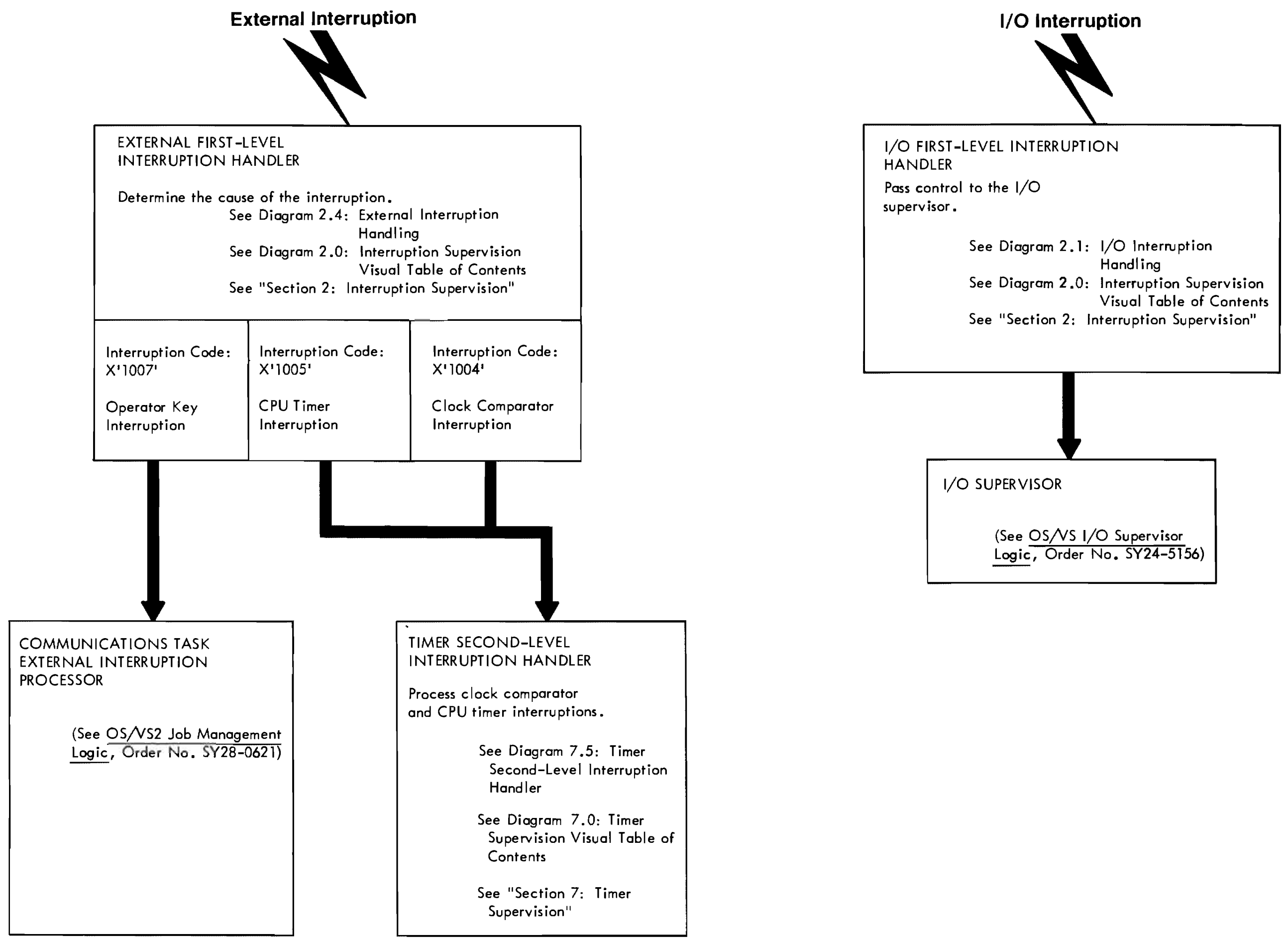
Select the next task to receive control.

See Diagram 3.17: Dispatcher

See Diagram 3.0: Task Supervision Visual Table of Contents

See "Section 3: Task Supervision"

Diagram 1.1 (Page 3 of 3)
Supervisor Overview and Visual Table of Contents for Diagrams



External Interruption

EXTERNAL FIRST-LEVEL INTERRUPTION HANDLER

Determine the cause of the interruption.
See Diagram 2.4: External Interruption Handling
See Diagram 2.0: Interruption Supervision Visual Table of Contents
See "Section 2: Interruption Supervision"

Interruption Code: X'1007'
Operator Key Interruption

Interruption Code: X'1005'
CPU Timer Interruption

Interruption Code: X'1004'
Clock Comparator Interruption

COMMUNICATIONS TASK EXTERNAL INTERRUPTION PROCESSOR

(See OS/VS2 Job Management Logic, Order No. SY28-0621)

TIMER SECOND-LEVEL INTERRUPTION HANDLER

Process clock comparator and CPU timer interruptions.

See Diagram 7.5: Timer Second-Level Interruption Handler

See Diagram 7.0: Timer Supervision Visual Table of Contents

See "Section 7: Timer Supervision"

I/O Interruption

I/O FIRST-LEVEL INTERRUPTION HANDLER

Pass control to the I/O supervisor.

See Diagram 2.1: I/O Interruption Handling
See Diagram 2.0: Interruption Supervision Visual Table of Contents
See "Section 2: Interruption Supervision"

I/O SUPERVISOR

(See OS/VS I/O Supervisor Logic, Order No. SY24-5156)

Diagram 1.2
Type-1 SVC Processing
Performed by the Supervisor

SVC FIRST-LEVEL INTERRUPTION HANDLER

SVC NUMBER	1	3	4	5	10	43	79
NAME OF ROUTINE	WAIT Routine	Exit Routine	GETMAIN Routines	FREEMAIN Routines	REGMAIN Routine	Stage 1 Exit Effector	Set Status Routine
DESCRIPTION	• Halt the execution of a program until a specified number of events have occurred.	• Handle exiting procedures for all programs except type-1 SVC routines.	• Allocate virtual storage in the requested amount from the subpool specified.	• Free allocated virtual storage in the length and subpool specified.	• Determine whether a GETMAIN or FREEMAIN macro instruction was issued, and pass control to either the GETMAIN routines (Diagram 6.2) or the FREEMAIN routines (Diagram 6.27).	• Create IRBs and TIRBs.	• Adjust a task's dispatchability.
DETAILED DIAGRAM	Diagram 3.7	Diagram 3.14	Diagram 6.2	Diagram 6.27	Diagram 6.2 and Diagram 6.27	Diagram 3.11	Diagram 3.18
OVERVIEW DIAGRAM	Diagram 3.1	Diagram 3.1	Diagram 6.1	Diagram 6.1	Diagram 6.1	Diagram 3.1	Diagram 3.1
TEXT INTRODUCTION	Section 3: Task Supervision	Section 3: Task Supervision	Section 6: Virtual Storage Supervision	Section 6: Virtual Storage Supervision	Section 6: Virtual Storage Supervision	Section 3: Task Supervision	Section 3: Task Supervision

SVC NUMBER	107	112	113	115	119
NAME OF ROUTINE	MODESET Routine	Release Routine (PGRLSE, nonsupervisor)	PGFIX/PGFREE/ PGLOAD/PGRLSE (supervisor)	Swap SVC Interface Routine	TESTAUTH Routine
DESCRIPTION	• Change the system mask, storage protection key, or the mode in the PSW.	• Make available all real and external page storage wholly associated with a specified area of virtual address space. • Return to the Type-1 Exit Routine or the caller.	• FIX: Fix virtual storage areas in real storage and make them ineligible for page-out. • FREE: Make previously fixed areas eligible for page-out. • LOAD: Bring specified areas of virtual storage into real storage from external page storage. • RELEASE: Branch to the Release routine (IEAPCLR).	• Obtain page control blocks (PCBs). • Place one PCB on the swap queue for further processing by IEAPSWAP and one on the queue for swap root PCBs.	• Determine whether a program is authorized to use the resource or function.
DETAILED DIAGRAM	Diagram 3.21	Diagram 5.30	Diagram 5.20	Diagram 5.61	Diagram 3.20
OVERVIEW DIAGRAM	Diagram 3.1	Diagram 5.4	Diagram 5.4	Diagram 5.4	Diagram 3.1
TEXT INTRODUCTION	Section 3: Task Supervision	Section 5: Paging Supervision	Section 5: Paging Supervision	Section 5: Paging Supervision	Section 3: Task Supervision

Diagram 1.3 (Page 1 of 4)
Types 2, 3, and 4 SVC Processing
Performed by the Supervisor

	SVC SECOND-LEVEL INTERRUPTION HANDLER			
SVC NUMBER	2 (Type 2)	6 (Type 2)	7 (Type 2)	8 (Type 2)
NAME OF ROUTINE	POST Routine	LINK Routine	XCTL Routine	LOAD Routine
DESCRIPTION	• Signal the occurrence of an awaited event to the waiting program.	• Pass control to a specified entry point. The load module is brought into virtual storage if a usable copy is not available. After the execution of the called program, control is returned to the instruction following the LINK macro instruction.	• Pass control to a specified entry point. The load module containing the entry point is brought into virtual storage if a usable copy is not available. No return is made to the program issuing the XCTL macro instruction.	• Bring the load module containing a specified entry point into virtual storage if a usable copy is not available.
DETAILED DIAGRAM	Diagram 3.8	Diagram 4.2	Diagram 4.8	Diagram 4.7
OVERVIEW DIAGRAM	Diagram 3.1	Diagram 4.1	Diagram 4.1	Diagram 4.1
TEXT INTRODUCTION	Section 3: Task Supervision	Section 4: Contents Supervision	Section 4: Contents Supervision	Section 4: Contents Supervision

Diagram 1.3 (Page 2 of 4)
Types 2, 3, and 4 SVC Processing
Performed by the Supervisor

	SVC SECOND-LEVEL INTERRUPTION HANDLER						
SVC NUMBER	9 (Type 2)	11 (Type 2)	12 (Type 2)	13 (Type 4)	14 (Type 2)	37 (Type 3)	40 (Type 2)
NAME OF ROUTINE	DELETE Routine	TIME Routine	SYNCH Routine	ABEND Routine	SPIE Routine	SEGLD or SEGWT Routine	EXTRACT Routine
DESCRIPTION	• Decrease the use count for modules fetched by a LOAD macro instruction. • Free storage occupied by a module that was requested through a LOAD macro instruction and is no longer needed.	• Determine the time of day and date and return both to the caller.	• Allow a supervisor routine to synchronously exit to a user program.	• Free control blocks, storage, and other resources used by the terminating task (or tasks) and subtasks. • Invoke ASIR if a STA user exit routine exists. • Invoke ABDUMP if requested.	• Perform the processing necessary to allow the user to specify an exit routine to be scheduled after a program interruption.	• Load specified overlay segments.	• Obtain information for a task from that task's TCB or the TCB of a subtask.
DETAILED DIAGRAM	Diagram 4.9	Diagram 7.2	Diagram 4.6	Diagrams 8.14 through 8.24	Diagram 3.6	Diagram 4.12	Diagram 3.4
OVERVIEW DIAGRAM	Diagram 4.1	Diagram 7.1	Diagram 4.1	Diagram 8.13	Diagram 3.1	Diagram 4.1	Diagram 3.1
TEXT INTRODUCTION	Section 4: Contents Supervision	Section 7: Timer Supervision	Section 4: Contents Supervision	Section 8: Termination	Section 3: Task Supervision	Section 4: Contents Supervision	Section 3: Task Supervision

Diagram 1.3 (Page 3 of 4)
Types 2, 3, and 4 SVC Processing
Performed by the Supervisor

	S V C S E C O N D - L E V E L I N T E R R U P T I O N H A N D L E R						
SVC NUMBER	41 (Type 3)	42 (Type 2)	44 (Type 2)	46 (Type 2)	47 (Type 2)	48 (Type 2)	51 (Type 4)
NAME OF ROUTINE	IDENTIFY Routine	ATTACH Routine	CHAP Routine	TTIMER Routine	STIMER Routine	DEQ Routine	SVCDUMP Routine
DESCRIPTION	• Inform the supervisor of a module's embedded entry-point name that was not established by the linkage editor.	• Create a task control block (TCB) to represent the task. • Schedule linkage to contents supervision to obtain the program to be first executed for the new task.	• Change the dispatching priority for a task or subtask.	• Calculate the time remaining in a requested interval. • Cancel the interval.	• Calculate and schedule a requested timed interval.	• Remove requests for resources from the resource queues.	• Branch to ABDUMP to display control blocks and storage for the specified task. • Display the virtual storage specified by the key 0 issuer of the SVC 51 macro instruction.
DETAILED DIAGRAM	Diagram 4.10	Diagram 3.2	Diagram 3.3	Diagram 7.4	Diagram 7.3	Diagram 3.10	Diagram 8.26
OVERVIEW DIAGRAM	Diagram 4.1	Diagram 3.1	Diagram 3.1	Diagram 7.1	Diagram 7.1	Diagram 3.1	Diagram 8.25
TEXT INTRODUCTION	Section 4: Contents Supervision	Section 3: Task Supervision	Section 3: Task Supervision	Section 7: Timer Supervision	Section 7: Timer Supervision	Section 3: Task Supervision	Section 8: Termination

Diagram 1.3 (Page 4 of 4)
Types 2, 3, and 4 SVC Processing
Performed by the Supervisor

	SVC SECOND-LEVEL INTERRUPTION HANDLER						
SVC NUMBER	56 (Type 2)	60 (Type 3)	62 (Type 2)				
NAME OF ROUTINE	ENQ/RESERVE Routine	STA Services	DETACH Routine				
DESCRIPTION	• Queue requests for resources (data sets, records, programs, etc.) to control access.	• Create a STA control block containing the address of a user-written exit routine and a parameter list. ABEND invokes the ABEND/STA Interface routine to schedule the user exit routine upon abnormal termination.	• Remove the specified TCB from the task queue. • Free the TCB's storage space and the associated problem-program register save area.				
DETAILED DIAGRAM	Diagram 3.9	Diagrams 8.43 through 8.47	Diagram 3.5				
OVERVIEW DIAGRAM	Diagram 3.1	Diagram 8.42	Diagram 3.1				
TEXT INTRODUCTION	Section 3: Task Supervision	Section 8: Termination	Section 3: Task Supervision				

SECTION 2

Interruption Supervision

2

Any interruption, except a machine-check interruption, causes CPU control to be taken from the executing program and given to an interruption-handling routine of the supervisor. There are four interruption-handling routines in the supervisor, one for each type of interruption except the machine check. (A machine check causes control to be passed directly to the Machine Check Handler, a standard recovery program of the operating system.)

The five types of interruptions are:

- Input/output interruptions, which occur when an I/O operation terminates, when an I/O error occurs, or an I/O device is made ready.
- SVC interruptions, which occur when an SVC instruction is executed.
- External/timer interruptions, which occur when the interrupt key is pressed on the system operator's console, or a time interval expires.
- Program interruptions, which occur when a program attempts an invalid operation (for example, execution of a privileged instruction by a program in problem state), when a data error is detected (for example, overflow), or when an address translation exception, enable/ disable interruption, or event-monitoring interruption occurs.
- Machine-check interruptions, which occur when the CPU detects a hardware malfunction.

An interruption causes the current PSW to be saved as the old PSW, and the new PSW to be executed. The new PSW causes control to be passed to the appropriate interruption-handling routine.

In most cases, the interruption handler does not do the detailed processing itself; instead, it analyzes the interruption and routes control to an interruption processing routine.

For input/output interruptions, the interruption handler branches to the Input/ Output Supervisor which performs input/ output services and handles input/output errors.

For SVC interruptions, the interruption handler determines which SVC service is required, causes the SVC routine to be brought into real storage (if necessary), and passes control to it.

For external/timer interruptions, the interruption handler determines the cause of the interruption and branches either to a timer service routine or to an external service routine.

For program interruptions, the interruption handler determines the cause of the interruption and does one or more of the following:

- Branches to the DSS Program Interruption Handler if DSS (the dynamic support system) is active (see the OS/VS Dynamic Support System Logic manual for more information on DSS operations).
- Branches to the paging supervisor for processing of address translation exceptions.
- Branches to a user error-handling routine.
- Branches to the generalized trace facility to record the event.
- Enables or disables interruptions.
- Terminates the task in which the interruption occurred.

The interruption handlers are disabled for all interruptions except machine checks so that they will not lose control before they have saved critical information about the interrupted program. This information consists of the register contents and the PSW information necessary to return control to the interrupted program after the interruption has been processed.

This section contains descriptions of I/O, SVC, external/timer, and program interruption handling. Machine-check handling is not described in this manual. For a description of machine-check handling, see the OS/VS Recovery Management Support Logic manual.

INPUT/OUTPUT INTERRUPTION HANDLING

(See Diagram 2.1)

When an input/output interruption occurs, the I/O FLIH (Input/Output First-Level Interruption Handler) is entered automatically as the result of the loading of the I/O new PSW.

The I/O FLIH branches to the input/output supervisor and, if necessary, the Page I/O Post routine. The input/output supervisor performs all input/output services and error handling for I/O interruptions while the Page I/O Post routine notes completion of paging I/O operations.

The I/O FLIH branches to the system management facility's Wait-time Collection routine if the system was in wait state at the time of the interruption. The I/O FLIH also interfaces with the system Trace routine to have the interruption recorded.

The I/O FLIH returns to either the interrupted program or the dispatcher depending on whether system services are required as a result of the I/O interruption.

SVC INTERRUPTION HANDLING

(See Diagrams 2.2 and 2.3)

When a system or user program executes a macro instruction, the last machine instruction of the resulting macro-expansion is often an SVC instruction. When executed, the SVC instruction causes an SVC interruption. The part of the supervisor that receives immediate control is called the SVC FLIH (SVC First-Level Interruption Handler).

Saving the Status of the Interrupted Program

The current PSW, containing the address of the next executable instruction, is stored by the machine in location X'20' in lower real storage. The SVC FLIH saves the register contents and old PSW of the interrupted routine to permit control to be returned to the routine at its next executable instruction. The register contents are saved by the SVC FLIH in a special SVC save area in lower real storage, called IEASCSAV. Figure 2-1, parts A and B show the saving of the PSW and the register contents, respectively, by the machine and by the SVC FLIH.

The register contents and old PSW remain in lower real storage or are moved to other save areas, depending on the type of SVC routine to be executed. If a type-1 SVC routine is to be executed, the register contents and old PSW are left in lower real storage. If one of the other types of SVC routines is to be executed, the information is moved to other save areas.

The reason for this is that a type-1 SVC routine cannot issue an SVC instruction or a macro instruction that expands into an SVC instruction. In addition, during execution of type-1 SVC routines, interruptions are disabled, ensuring that an SVC instruction will not be issued in another routine. Consequently, there is no danger of a second entry to SVC processing during execution of a type-1 SVC routine and therefore no danger of having the information for the current interruption overlaid.

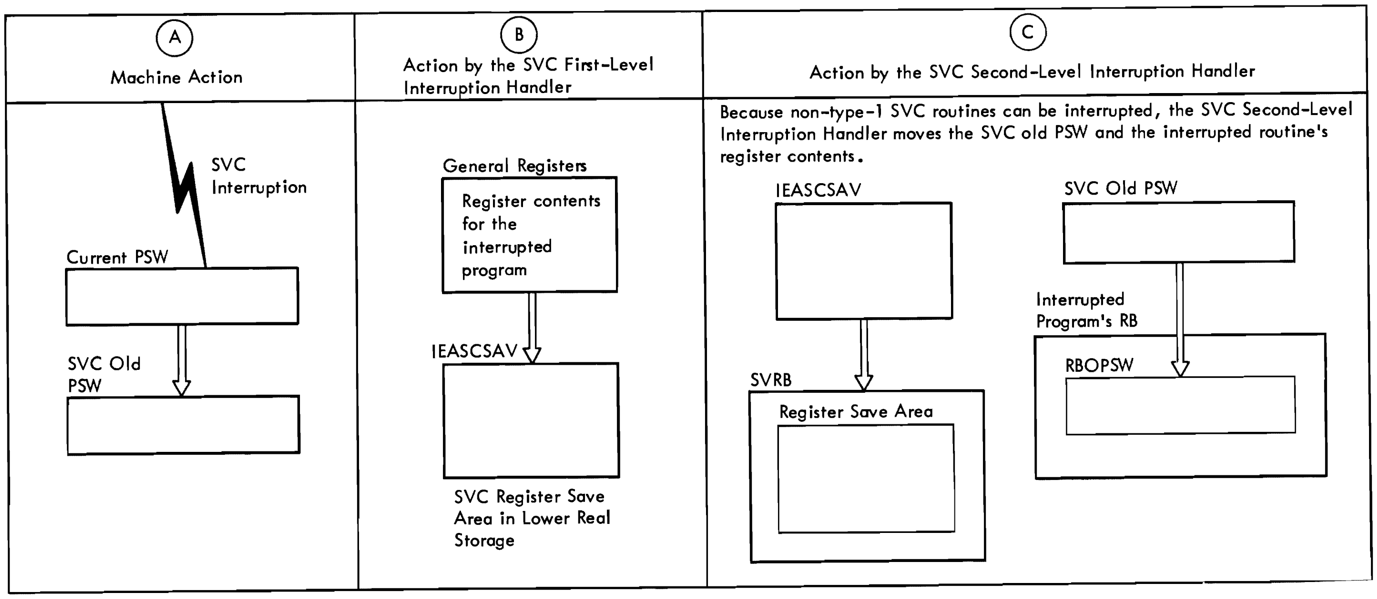

A
Machine Action
SVC Interruption
Current PSW
SVC Old PSW
B
Action by the SVC First-Level Interruption Handler
General Registers
Register contents for the interrupted program
IEASCSAV
SVC Register Save Area in Lower Real Storage
C
Action by the SVC Second-Level Interruption Handler
Because non-type-1 SVC routines can be interrupted, the SVC Second-Level Interruption Handler moves the SVC old PSW and the interrupted routine's register contents.
IEASCSAV
SVC Old PSW
Interrupted Program's RB
RBOPSW
SVRB
Register Save Area

Figure 2-1. Actions taken by the SVC First-Level and Second-Level Interruption Handlers to save program status information.

However, any of the other types of SVC routines can be interrupted by an SVC interruption. This requires that the status information for the current interruption be moved to new areas to prevent it from being lost.

To determine the type of SVC routine that will be executed, the SVC FLIH examines the SVC table. There are two parts to this table, one containing entries for user-supplied SVC routines, the other containing entries for IBM-supplied SVC routines. The number and type of routines in the two parts depend on the particular system that the user specified at system generation. There is one entry in the SVC table for each SVC routine in the generated system. Each entry contains a code showing the SVC type.

If the SVC table indicates that a type-1 routine is to be executed, the SVC FLIH branches to the routine, and the status information for the current interruption is left in lower real storage.

But, if the SVC routine is not type-1, and is interruptable, the SVC FLIH branches to the SVC SLIH (SVC Second-Level Interruption Handler) to create an SVRB (supervisor request block). The SVRB is used to hold the interrupted routine's register contents and return information for the SVC routine. The SVC SLIH moves the SVC old PSW, the ILC (instruction length code), and the interruption code to the interrupted routine's RB. The register contents are moved to the SVRB instead of the RB because the RB may lack a register save area. The SVRB is constructed by using the quickcell allocation technique described in "Section 6: Virtual Storage Supervision."

The SVC SLIH also sets status bits in the SVRB. These bits indicate that the SVRB is a dynamic request block that can be freed by the supervisor's Exit routine and that no attention exits can be taken while the SVRB is active. The SVC SLIH then adds the SVRB to the head of the RB queue belonging to the interrupted routine's TCB. (For an explanation of the RB queue, see "Section 3: Task Supervision.")

Determining Whether the Disabled SVC Routine is in Real Storage

After the SVRB has been initialized and added to the RB queue, the SVC SLIH determines whether the SVC routine is to receive control disabled, and if so, whether it is in real storage. To do this, it tests the "type" bits in the SVC table entry passed by the SVC First-Level Interruption Handler. Type-2 SVC routines are located in the nucleus and therefore are fixed in real storage. Type-3 and type-4 SVC routines that receive control disabled might not be in real storage, so the SVC SLIH issues a Load Real Address (LRA) instruction and tests the condition code to determine whether the SVC routine is present.

Enabling and Disabling Interruptions

If the SVC routine is in real storage and is to receive control with interruptions disabled, the SVC SLIH branches directly to it since the SVC SLIH itself is already disabled.

If the SVC routine is to receive control with interruptions enabled, the SVC SLIH branches to MODESET (a centralized facility to set the system mask -- see "Section 3: Task Supervision"). MODESET changes the system mask to enable interruptions as requested, and then returns control to the SVC SLIH. Upon return from MODESET, the SVC SLIH restores general registers 0, 1, 13, and 15 from the SVRB register save area so that they contain the same values as when the SVC was issued, and then branches to the SVC routine.

When a type-3 or type-4 SVC routine to be entered disabled is not in real storage, the SVC SLIH branches to MODESET to enable interruptions. While enabled, the SVC SLIH references the first instruction of the SVC routine, causing the SVC routine to be paged in by the paging supervisor. When the SVC routine has been paged in, the SVC SLIH issues a MODESET SVC (SVC 107) to disable interruptions. When control is returned from MODESET, the SVC SLIH restores registers 0, 1, 13, and 15 from the SVRB general register save area so they will have the same values as when the SVC was issued, and branches to the SVC routine.

Minor Functions of SVC Interruption Handling

In addition to the major functions already described, the SVC interruption handlers perform five minor functions:

1. They use the TESTAUTH routine to verify that the issuer is authorized to use an SVC routine that requires authorization. (If the user is not authorized, the task issuing the SVC is scheduled for abnormal termination.)

2. They check the validity of the SVC number.

3. They allow the interruption to be recorded.

4. They make available the addresses of three important control blocks (CVT,

TCB, and RB) for use by other supervisor routines during later processing of the SVC interruption.

5. They pass the address of the appropriate exit routine in the return register so that the SVC routine, when complete, can return control by means of a simple branch, without the need for tests.

TIMER/EXTERNAL INTERRUPTION HANDLING

(See Diagram 2.4.)

The External FLIH (External First-Level Interruption Handler) receives control after an external interruption. It saves the contents of the general registers of the interrupted program in the current TCB. If the system wait TCB is the current TCB, the External FLIH passes control to the system management facility's Wait-Time Collection routine. This routine returns control to the External FLIH.

The External FLIH saves the external old PSW and the interruption code in the current RB. It then provides for tracing by branching to CVTTRACE and by issuing a Monitor Call instruction to have the external interruption recorded by the generalized trace facility.

The External FLIH determines the cause of the interruption from the interruption code. It then passes control to the Timer Second-Level Interruption Handler if the interruption was caused by the clock comparator or CPU timer, or to the communications task's External Interruption Handler if it is an operator key interruption. On return from the Timer SLIH or the communications task's External Interruption Handler, the External FLIH stores the time-of-day (TOD) clock value into a special save area within the system management facility's EXCP Counter routine.

The TOD clock value must be stored as a precaution for the case in which the system wait TCB is the current TCB, and a clock comparator or CPU timer interruption occurs but no task switch occurs. In this case, if the TOD clock value in the EXCP Counter routine had not been reset by the External FLIH, the calculated wait time would include execution time for the external interruption handlers.

Only clock comparator, CPU timer, and interrupt key interruptions are processed. All other external interruptions are ignored.

The External FLIH then branches to the dispatcher, which determines the next task to receive control.

PROGRAM INTERRUPTION HANDLING

(See Diagrams 2.5 -- 2.9)

When a program interruption occurs, an interruption code describing the cause of the interruption is stored at location X'8E'. There are four types of program interruptions; program-check interruptions, address translation exceptions, enable/disable interruptions, and event monitoring interruptions. The following are brief descriptions of these types of program interruptions.

- Program-check interruptions: These interruptions (indicated by program interruption codes X'01' through X'0F') are caused by invalid actions such as using incorrect addresses, issuing invalid operation codes, or issuing privileged instructions. Additional causes of program-check interruptions are fixed-point overflow, decimal overflow, exponent underflow, and loss of significance. These interruptions can be masked out.

- Address translation exceptions: These interruptions (indicated by interruption codes X'10' through X'12') are caused by a segment translation exception, a page translation exception (also called a missing page interruption), or a translation specification exception. A page translation exception (page fault) is the most frequent one and generally occurs when a program tries to access a virtual storage address in a page that has not yet been brought into real storage.

- Enable/disable interruptions: These interruptions (indicated by interruption code X'13') are caused by a program's issuing a Set System Mask (SSM) instruction to alter the interruption state of the system.

- Event monitoring interruptions: These interruptions (indicated by interruption codes X'40' and X'80') are caused by the PER (program event recording) feature or by a Monitor Call (MC) instruction.

Interruptions other than the PER interruption are mutually exclusive. Multiple interruption conditions can occur concurrently since the PER interruption can occur simultaneously with each of the other kinds of program interruption.

The Program Interruption Handler is given control automatically after any program interruption. Upon entry, all interruptions, except machine checks, are disabled and dynamic address translation is inhibited to prevent translation specification recursions. After saving the status information for the interrupted program, the Program Interruption Handler determines the cause of the interruption by examining the interruption code at location X'8E'. It then processes the interruption, handling translation specification exceptions first, invalid recursions next, and then page transalation exceptions and the other causes of program interruptions other than translation specification exceptions, the Program Interruption Handler enables itself to receive dynamic address translation exceptions.

PROCESSING FOR PARTICULAR PROGRAM INTERRUPTIONS

The following subsections provide more detailed information on particular types of program interruptions and how they are processed.

Translation Specification Exceptions

(See Diagram 2.5)

A translation specification exception is an interruption that occurs when a page table entry, segment table entry, or the control register pointing to the segment table contains information in an invalid format. Translation specification exceptions are indicated by program interruption code X'12'.

For a translation specification interruption, the Program Interruption Handler branches to the paging supervisor's error routine (IEAPSER). If the interruption occurred in a system module, indicated by protection key 0, the Program Interruption Handler branches to the paging supervisor with register 0 set to indicate major trouble. When major trouble is indicated, the paging supervisor puts the system in a wait state. If the trouble is minor, the paging supervisor error routine calls ABTERM to schedule termination of the specified task and returns control to the Program Interruption Handler, which records the interruption via the Trace routine and the generalized trace facility. The Program Interruption Handler then exits to the dispatcher to allow the task to be abnormally terminated.

Recursion

(See Diagram 2.5)

Recursion is not a specific type of interruption, it is an unanticipated reentrance into the Program Interruption Handler that occurs when the Program Interruption Handler or one of its extensions causes another program interruption.

A valid recursion can occur in the resident GTF (generalized trace facility) extension of the Program Interruption Handler and is resolved by the GTF extension, which reestablishes the original program interruption conditions. The GTF extension resaves the low storage address save area, control registers, old PSW, and interruption code, and uses these to reestablish the information required by the Program Interruption Handler when a valid recursion occurs.

Recursions that occur when the GTF extension is not executing are treated as errors, and the task that was executing is abnormally terminated.

Event Recording

(See Diagram 2.5)

Event recording occurs as a result of a PER or MC (Monitor Call) program interruption, or when the Program Interruption Handler processes any interruption for which the tracing option is in effect.

For MC program interruptions, the GTF resident routine is used. This routine returns control directly to the interrupted program. No event recording is performed for invalid recursions.

PER interruptions are serviced by the DSS (dynamic support system) Program Interruption Handler and its related routines (see the OS/VS Dynamic Support System Logic manual).

For interruptions consisting of only MC or PER, the supervisor's Program Interruption Handler exits to the interrupted program via LPSW, unless a task switch has been established by GTF, in which case GTF passes control to the dispatcher.

Page Translation Exceptions

(See Diagrams 2.5 and 2.7)

A page translation exception (also called a missing page interruption) occurs when a virtual address cannot be translated because the invalid bit in the page table entry for the address is set. Page translation exceptions (page faults) are signified by program interruption code X'11'. A page fault can also be caused by referencing an invalid page; this condition is treated as a protection exception program check (see Diagrams 2.5 and 2.8).

When a page fault occurs in the input/output supervisor while the "IOS-in-process" switch is set, the current task is abnormally terminated via the ABTERM Prologue routine. If a page fault occurs in a disabled routine and the system lock is on, the task is also scheduled for abnormal termination by the ABTERM Prologue routine. If the interruption occurs in a disabled routine and the system lock is not on, the Program Interruption Handler turns it on (until the referenced page is available) to prohibit the execution of tasks that run with interruptions disabled. Only tasks that run with interruptions enabled are dispatched until the system lock is turned off by the dispatcher.

When a page fault occurs in a user task and the user has a valid SPIE exit routine, the exit routine is dispatched. Otherwise, to process a page fault, the Program Interruption Handler branches to the paging supervisor's Program-Check Interruption Extension (IEAPIX). The extension either reclaims the referenced page, assigns the page for the first time, or initiates a paging operation. The Program-Check Interruption Extension returns to the Program Interruption Handler indicating whether the page was reclaimed, assigned, or is to be paged in. The Program Interruption Handler exits directly to the task if the page was reclaimed or assigned, or to the dispatcher to perform a task switch.

When an invalid page has been referenced, the Program Interruption Handler changes the interruption code to a protection exception and handles it as a normal program-check interruption.

Segment Translation Exceptions

(See Diagrams 2.5 and 2.8)

When a segment address cannot be translated by the hardware because the segment table address is invalid, or the "segment-invalid" bit in the segment table entry is set, a segment translation exception occurs. Segment translation exceptions are signified by program interruption code X'10'.

When a segment translation exception occurs, the Program Interruption Handler treats it the same as it treats a page translation exception when an invalid page has been referenced: the Program Interruption Handler changes the interruption code to a protection exception and handles it as a normal program-check interruption.

Program Interruptions from SSM (Set System Mask) Instructions

(See Diagrams 2.5 and 2.6)

A centralized facility, MODESET, is used to set the system mask to enable or disable interruptions. Any attempt to change the system mask with an SSM instruction causes a program interruption (interruption code X'13').

To prevent a recursion that would result from a page translation interruption, the Program Interruption Handler issues a Load Real Address instruction to determine if the SSM operand is in real storage. If the operand is not in real storage, the interruption handler uses the paging supervisor's Program-Check Interruption Extension (IEAPIX1) to obtain the page containing the operand. When the SSM operand is in real storage, the state of the supervisor lock determines if and when the Program Interruption Handler must set the system mask as indicated by the SSM operand.

If the state indicated by the SSM instruction is the same as the state of the requester, the interruption is treated as a NOP and control is returned to the interrupted routine by using an LPSW instruction.

If the supervisor lock is set and the issuer of the SSM instruction is the paging supervisor, the Program Interruption Handler sets the system mask as indicated by the SSM operand. If the supervisor lock is set and the issuer of the SSM instruction is not the paging supervisor, the Program Interruption Handler stacks the SSM instruction, (decrements the address in the old PSW by four bytes and stores it in the RB, saves registers in the TCB, and flags the RB nondispatchable until the supervisor lock is off). It indicates a task switch if necessary (if IEATCBP=IEATCBP+4) by setting IEATCBP to zero. If a task switch has been scheduled, the Program Interruption Handler exits to the dispatcher; otherwise, it returns control to the user via an LPSW.

Program-Check Interruptions

(See Diagrams 2.5 and 2.8)

The Program Interruption Handler receives control automatically after any program-check interruption. It tests the old PSW to determine whether the interruption occurred in the supervisor or in user code.

If the interruption occurred in a routine in the supervisor state, control is passed to the ABTERM Prologue routine which schedules abnormal termination of the task that was interrupted.

If the interruption occurred in user code, the Program Interruption Handler tests the TCB for a PIE (program interruption element) address. A PIE is a control block associated with a user's error-handling routine. If the user anticipates a program-check interruption and wishes to perform his own error handling, he issues a SPIE (set program interruption element) macro instruction. In response to the SPIE macro instruction, the SPIE service routine constructs a PIE and inserts its address in the TCB.

If the supervisor finds no PIE address in the TCB (indicating that the user did not specify a routine to process program interruptions), the ABTERM Prologue routine is entered. If a PIE address is found, the Program Interruption Handler tests to determine whether the PIE is in real storage. If a PIE exists and is not in real storage, the Program Interruption Handler uses the Program-Check Interruption Extension of the paging supervisor to page in the control block.

The Program Interruption Handler then tests the "in-use" bit in the PIE. This bit is set whenever control is given to the user's error-handling routine. If the bit is on, it means that the current program-check interruption occurred while the error routine was processing a previous program-check interruption, and the task must be terminated. If the "in-use" bit is zero, the Program Interruption Handler determines whether the PICA is in real storage and if not, uses the paging supervisor's Program-Check Interruption Extension to page in the control block. Then the Program Interruption Handler determines whether the kind of program interruption that occurred was specified in the SPIE macro instruction. If it was, control is passed to the user's error-handling routine. If it was not, control is passed to the ABTERM Prologue routine which, with the ABTERM routine, schedules the abnormal termination of the task.

When a user error-handling routine completes its processing, the supervisor Exit routine is entered, and the interrupted program regains control via the dispatcher.

TRACING EVENTS

(See Diagrams 2.10 and 2.11)

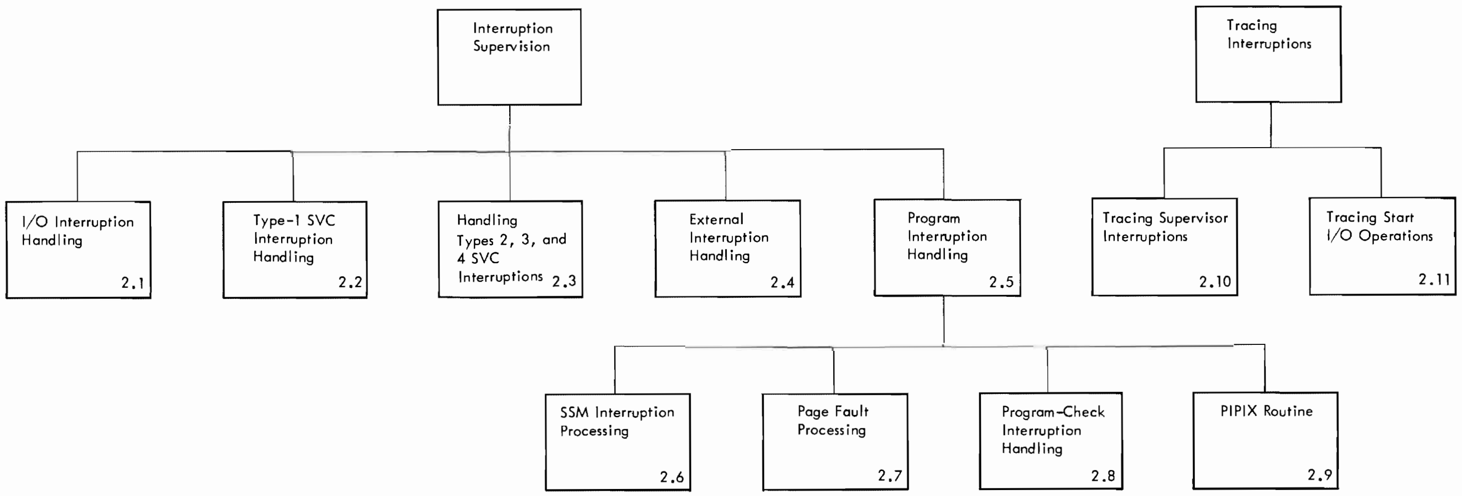
The Trace function records up to 32 bytes of information, in the form of a Trace table entry, each time it is called. The information it records depends upon which of the six entry points is used. For each entry point, Trace processing is tailored to a specific caller as follows:

Caller	Trace Information Recorded
Dispatcher	PSW to be dispatched; registers 0, 1, and 15 of the task to be dispatched; TCB to be dispatched, time stamp.
External FLIH	Old PSW; registers 0, 1, and 15; timer queue element address (if a timer interruption occurred); time stamp.
I/O FLIH	Old PSW; CSW; time stamp.
IOS Start I/O	Condition code; device address; CAW; CSW; TCB associated with I/O; time stamp.
Program FLIH	Old PSW; registers 0, 1, and 15; page or segment exception address; current TCB; time stamp.
SVC FLIH	Old PSW; registers 0, 1, and 15; current TCB; time stamp.

The tracing option is selected at nucleus initialization time. If the tracing option is selected (that is, the number of Trace entries > 0) the nucleus initialization program sets CVTTRACE = BR 10; otherwise, CVTTRACE is set = BR 11 (to return directly to the caller, bypassing the tracing function).

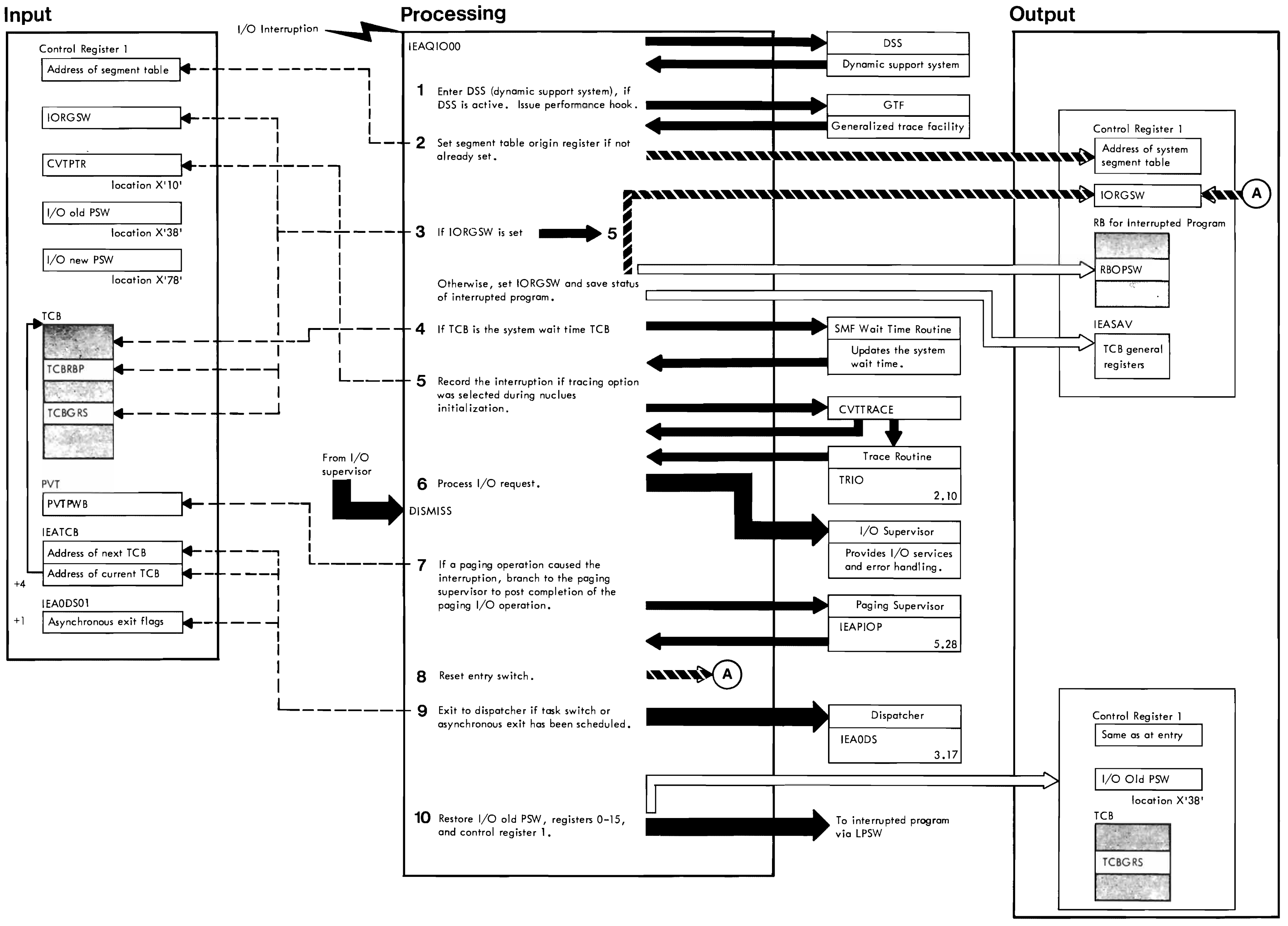
A dump of the trace table can be obtained via SVC DUMP or ABDUMP.

• **Diagram 2.0**
Interruption Supervision
Visual Table of Contents



Interruption Supervision
I/O Interruption Handling 2.1
Type-1 SVC Interruption Handling 2.2
Handling Types 2, 3, and 4 SVC Interruptions 2.3
External Interruption Handling 2.4
Program Interruption Handling 2.5
SSM Interruption Processing 2.6
Page Fault Processing 2.7
Program-Check Interruption Handling 2.8
PIPIX Routine 2.9
Tracing Interruptions
Tracing Supervisor Interruptions 2.10
Tracing Start I/O Operations 2.11

Diagram 2.1
I/O Interruption Handling



1 Enter DSS (dynamic support system), if DSS is active. Issue performance hook.

2 Set segment table origin register if not already set.

3 If IORGSW is set → 5

Otherwise, set IORGSW and save status of interrupted program.

4 If TCB is the system wait time TCB

5 Record the interruption if tracing option was selected during nuclues initialization.

6 Process I/O request.

7 If a paging operation caused the interruption, branch to the paging supervisor to post completion of the paging I/O operation.

8 Reset entry switch.

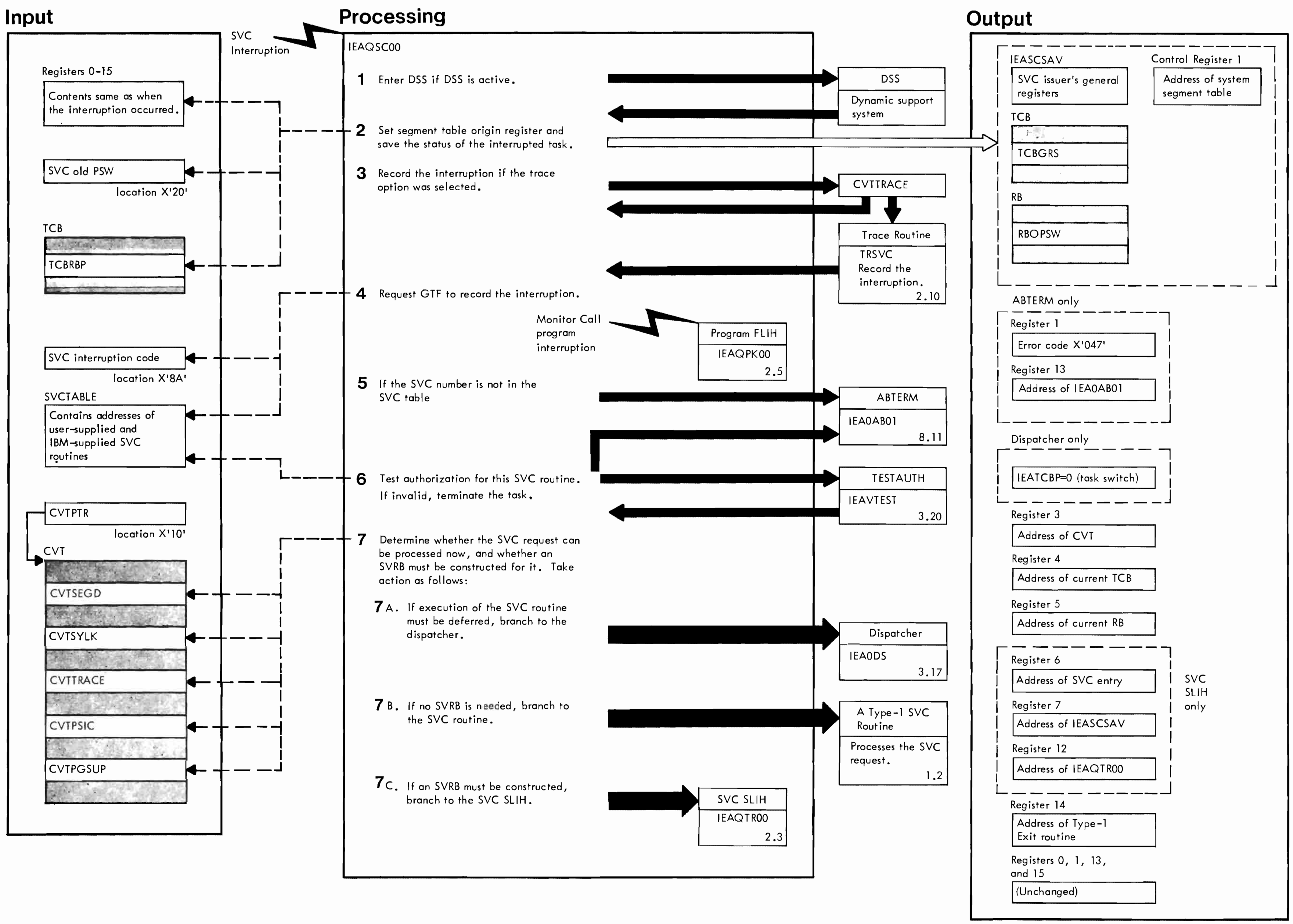
9 Exit to dispatcher if task switch or asynchronous exit has been scheduled.

10 Restore I/O old PSW, registers 0-15, and control register 1.

Diagram 2.1 I/O Interruption Handling (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
3 When the I/O First-Level Interruption Handler (I/O FLIH) is entered initially, it sets the I/O original entry switch (IORGSW). This switch permits the I/O FLIH to distinguish between an initial entry for an I/O operation (when register contents and status information must be saved) and recursions like those resulting from retries of the I/O operation due to errors (when register contents have already been saved).	IEAQI000	
4	IEAQWAIT	
5 The I/O FLIH branches to CVTTRACE which contains either a BR 11 or a branch instruction to the Trace routine (if the tracing option was selected). Control is returned to the I/O FLIH.	IEAQIO00 IEAQTRCE	NIOS TRIO
6	IEAQIO00 DISMISS	
7	IEAPIOP	
8	IEAQIO00	OFFIOSW
9 A task switch is pending if IEATCBP / IEATCBP+4. An asynchronous exit routine is pending if IEA0DS01+1 = X'F0'.	IEAQIO00	IODISP
10 The PSW saved in the top RB (RBOPSW) of the current TCB (IEATCBP+4) is restored to the I/O old PSW location, control register 1 is set to the appropriate segment table, general registers are restored, an MC instruction is issued via a HOOK macro instruction, and control is passed to the interrupted program via LPSW.	IEAQIO00	

Diagram 2.2
Type-1 SVC Interruption Handling



Input

Registers 0-15
Contents same as when the interruption occurred.

SVC old PSW
location X'20'

TCB
TCBRBP

SVC interruption code
location X'8A'

SVCTABLE
Contains addresses of user-supplied and IBM-supplied SVC routines

CVTPTR
location X'10'

CVT
CVTSEGD
CVTSYLK
CVTTRACE
CVTPSIC
CVTPGSUP

SVC Interruption

Processing

IEAQSC00

1 Enter DSS if DSS is active.

2 Set segment table origin register and save the status of the interrupted task.

3 Record the interruption if the trace option was selected.

4 Request GTF to record the interruption.

Monitor Call program interruption

5 If the SVC number is not in the SVC table

6 Test authorization for this SVC routine. If invalid, terminate the task.

7 Determine whether the SVC request can be processed now, and whether an SVRB must be constructed for it. Take action as follows:

7A. If execution of the SVC routine must be deferred, branch to the dispatcher.

7B. If no SVRB is needed, branch to the SVC routine.

7C. If an SVRB must be constructed, branch to the SVC SLIH.

DSS
Dynamic support system

CVTTRACE

Trace Routine
TRSVC
Record the interruption.
2.10

Program FLIH
IEAQPK00
2.5

ABTERM
IEA0AB01
8.11

TESTAUTH
IEAVTEST
3.20

Dispatcher
IEA0DS
3.17

A Type-1 SVC Routine
Processes the SVC request.
1.2

SVC SLIH
IEAQTR00
2.3

Output

IEASCSAV
SVC issuer's general registers

Control Register 1
Address of system segment table

TCB
TCBGRS

RB
RBOPSW

ABTERM only
Register 1
Error code X'047'
Register 13
Address of IEA0AB01

Dispatcher only
IEATCBP=0 (task switch)

Register 3
Address of CVT
Register 4
Address of current TCB
Register 5
Address of current RB

Register 6
Address of SVC entry
Register 7
Address of IEASCSAV
Register 12
Address of IEAQTR00
SVC SLIH only

Register 14
Address of Type-1 Exit routine

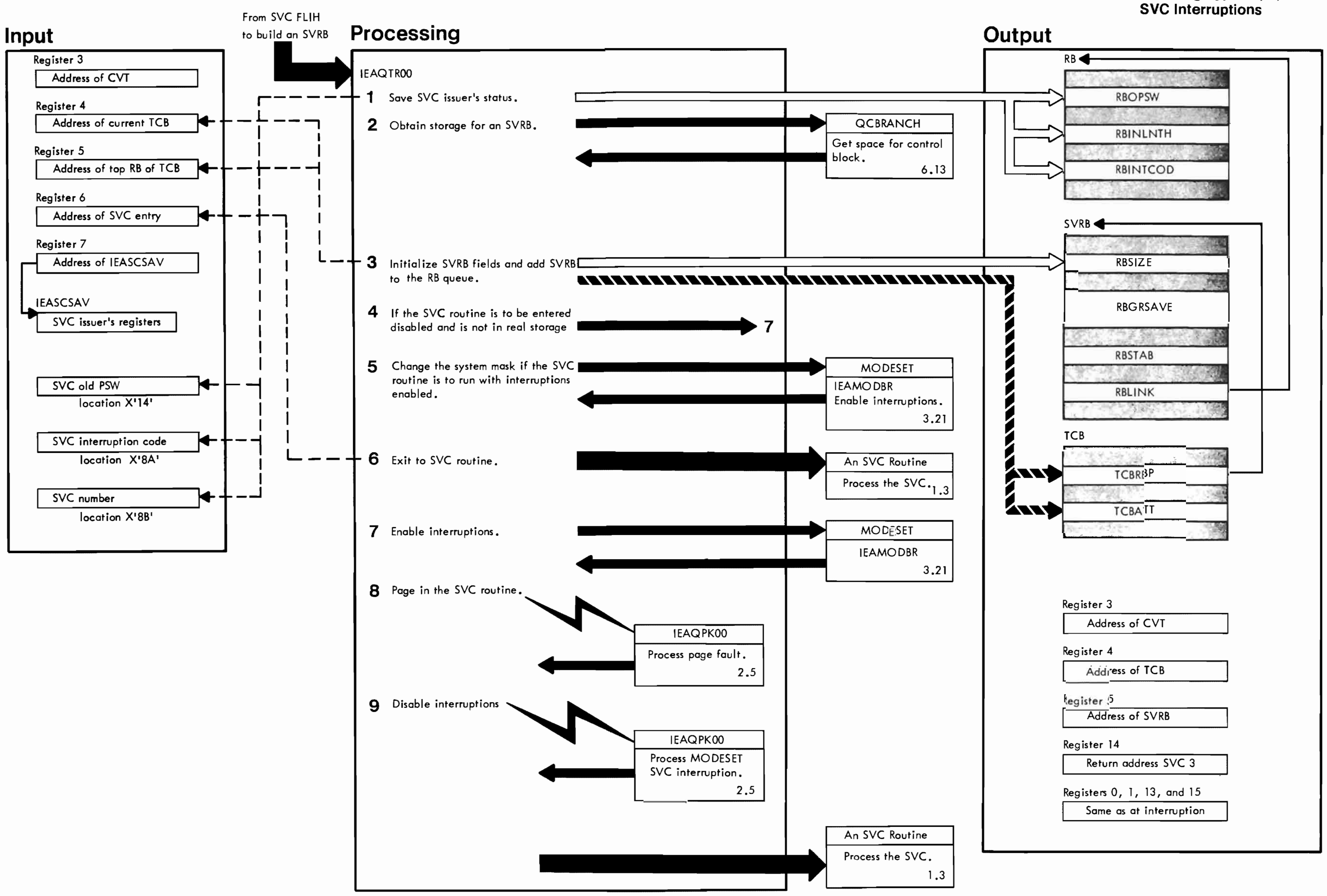
Registers 0, 1, 13, and 15
(Unchanged)

Diagram 2.2 Type-1 SVC Interruption Handling (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
1 The segment table register is set to the address of the system segment table if it is not already set.	IEAQSC00	
3 The SVC First-Level Interruption Handler (SVC FLIH) branches to CVTTRACE which contains either a BR 11 (return) or a BR 10 to TRSVC (if the trace option was selected). Control is returned to the SVC FLIH.	IEAQTRCE	TRSVC
4 A Monitor Call instruction is issued (via the HOOK macro instruction) for the SVC interruption class. If event monitoring is in effect for the class, a program interruption results. (See Diagram 2.5.)		
5 The SVC FLIH compares the SVC number (in the SVC interruption code) with the SVC table to see if the number is valid for this system. When an SVC number is less than the lowest user SVC number in the system or higher than the highest IBM SVC number, the SVC FLIH branches to ABTERM to schedule abnormal termination of the task.	IEAQSC00	IGCERROR
6 The SVC FLIH inspects the function code in the APF table. If zero, processing continues at Step 7. Otherwise, the SVC FLIH checks whether the current task is in supervisor state or key 0. If it is, processing continues at Step 7. If it is not, the SVC FLIH branches to the task supervision TESTAUTH routine to determine whether the SVC issuer is authorized for the requested functions. If TESTAUTH determines that the SVC issuer is not authorized for that SVC, the SVC FLIH branches to ABTERM to schedule abnormal termination of the task. If the SVC issuer is properly authorized, processing continues at Step 7.	IEAVTEST	
7 The SVC FLIH checks control block fields to determine its next action, as shown in the following table:	IEAQSC00	

Step	Supervisor Lock Set? (CVTSYSLK=FF)	SVC Issued by Paging Supervisor? (CVTPSIC=1 or IEATCB+4=a Paging Sup. TCB) (CVTPGSUP)	Type-1 SVC? (IGCTYPE=0)	Interruptions Enabled in SVC? (IGCABLE=1)	Action Taken by SVC FLIH
7A	yes	no	-	no	Backs-up PSW in RB so the SVC instruction will reexecute when the supervisor lock is reset, and branches to the dispatcher.
7B	no	-	yes	-	Branches to Type-1 SVC routine.
	yes	yes	yes	-	
7C	no	-	no	-	Branches to SVC SLIH to construct an SVRB and to start the SVC routine.
	yes	yes	no	-	
	yes	-	no	yes	

Diagram 2.3
Handling Types 2, 3, and 4
SVC Interruptions

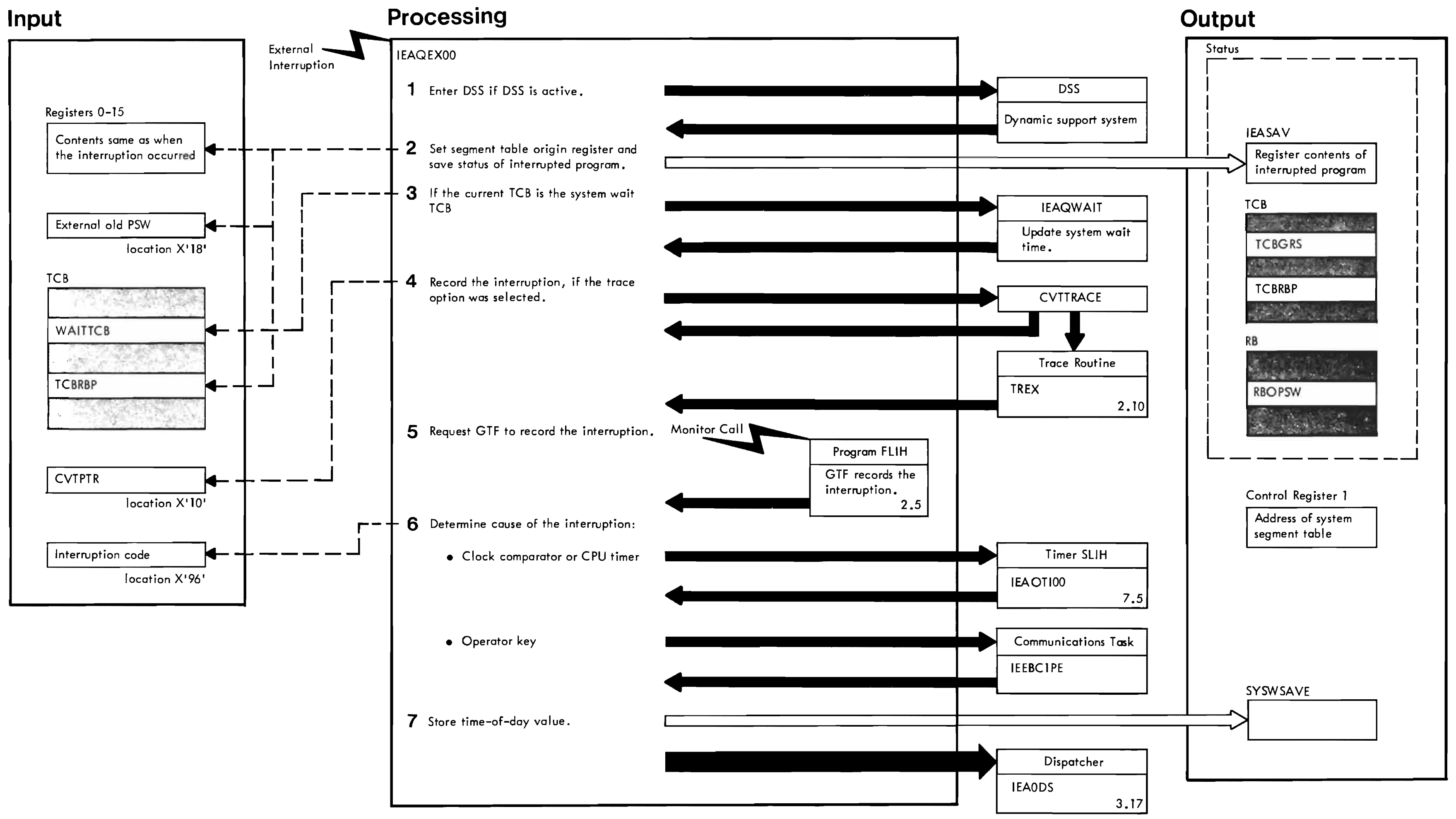


1 Save SVC issuer's status.
2 Obtain storage for an SVRB.
3 Initialize SVRB fields and add SVRB to the RB queue.
4 If the SVC routine is to be entered disabled and is not in real storage
5 Change the system mask if the SVC routine is to run with interruptions enabled.
6 Exit to SVC routine.
7 Enable interruptions.
8 Page in the SVC routine.
9 Disable interruptions

Diagram 2.3 Handling Types 2, 3, and 4 SVC Interruptions (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
3 Since the RB may not have a register save area, the SVC SLIH moves the registers of the interrupted program to the save area of the SVRB (REGRSAVE). Hereafter, if the currently requested SVC routine causes a new SVC interruption, its register contents and old PSW will be stored in lower real storage without causing the original SVC issuer's status to be lost. The SVC SLIH sets RBSIZE to the number of doublewords in the SVRB, and sets fields in RBSTAB and RBSTAB2 to indicate the RB is a dynamic SVRB (this tells the Exit routine that the SVRB can be freed). If the SVC is type-3 or type-4, the SVC SLIH sets field RSTAB1 bit RBTRSVRB. This bit is a signal to contents supervision routines that, for loads of an SVC subsequent to the first load, contents supervision must fill the RBCDE field with a pointer to the CDE for that load. The RBCDE pointer will be used by ABDUMP to determine the address and size of the SVC module to include in the printed dump. This indicator is necessary, since, for loads of all SVC routines, the SVC interruption handlers branch directly to the virtual address of the SVC routine as indicated in the SVC table (no CDEs are examined and no RBCDE pointer is filled in). Without special processing, ABDUMP would be unable to distinguish between first and subsequent loads of an SVC routine and would in every case print out the first load. The SVC SLIH stores the SVRB address into TCBRBP and stores the previous top RB address into RBLINK. Then, it sets TCBATT=X'20' to prevent attention exits from being executed while the SVC routine is running. Then, the SVC SLIH sets the first word of the RB old PSW equal to the first word of the SVC new PSW to indicate that the SVC routine is key 0, supervisor state, with translation enabled.	IEAQTR00	
8 While enabled, the SVC SLIH refers to the first instruction of the SVC routine. This produces a program interruption that causes the SVC routine to be paged into real storage. The SVC SLIH regains control when the paging operation has been completed.		
9 If the SVC routine is to be entered disabled, the SVC SLIH issues a MODESET SVC. If the system lock was set while paging was in process, the MODESET SVC is deferred by the SVC FLIH. The page containing the SVC routine must be retested to ensure that it is still in real storage. If it is not, the page-in function (Step 7) is repeated.		

Diagram 2.4
External Interruption Handling



1 Enter DSS if DSS is active.

2 Set segment table origin register and save status of interrupted program.

3 If the current TCB is the system wait TCB

4 Record the interruption, if the trace option was selected.

5 Request GTF to record the interruption.

6 Determine cause of the interruption:

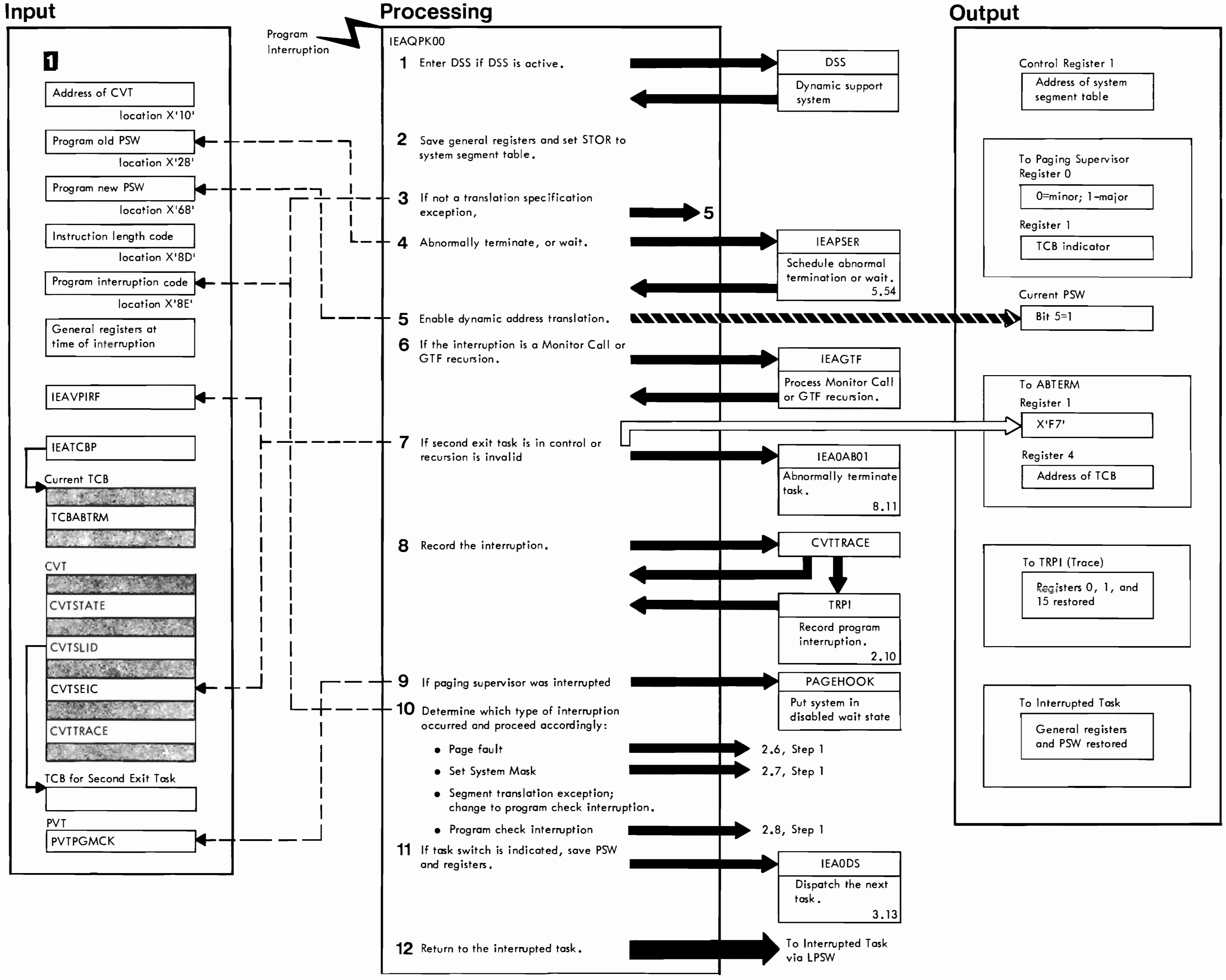
- Clock comparator or CPU timer
- Operator key

7 Store time-of-day value.

Diagram 2.4 External Interruption Handling (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
2 The External First-Level Interruption Handler (External FLIH) first stores registers 14 and 15 in IEASAV; then it uses these registers to store registers 0 through 15 into the TCBGRS field of the TCB whose address is in IEATCBP+4. It then stores the external old PSW in the current RB (RBOPSW).		
3	IEAQWAIT	
4 The External FLIH branches to CVTTRACE which contains either a BR 11 (return) or a BR 10 instruction to the Trace routine (if tracing was selected during NIP). Control is returned to the External FLIH.	IEAQEX00 IEAQTRCE	EXTRACE TREX
5 A Monitor Call (MC) instruction is issued via a HOOK macro instruction. The macro expansion generates the MC instruction with the class set to external interruption. This causes a program interruption if event monitoring for this class is in effect. (See Diagram 2.5.)	IEAPSER	
6 The External FLIH branches to the Timer SLIH if the interruption code is X'1004' (clock comparator) or if X'1005' (CPU timer). It branches to the communications task external interruption handler if byte 135, bit 1, is one. Interruption codes other than these are ignored.	IEA0TI00 IEEBC1PE	
7 The External FLIH stores the TOD clock value in SYSWSAVE so that the time required to process this interruption will not be charged to the WAIT TCB.	IEAVEX00 IEA0DS	

• **Diagram 2.5**
Program Interruption Handling



1 Enter DSS if DSS is active.

2 Save general registers and set STOR to system segment table.

3 If not a translation specification exception,

4 Abnormally terminate, or wait.

5 Enable dynamic address translation.

6 If the interruption is a Monitor Call or GTF recursion.

7 If second exit task is in control or recursion is invalid

8 Record the interruption.

9 If paging supervisor was interrupted

10 Determine which type of interruption occurred and proceed accordingly:

- Page fault
- Set System Mask
- Segment translation exception; change to program check interruption.
- Program check interruption

11 If task switch is indicated, save PSW and registers.

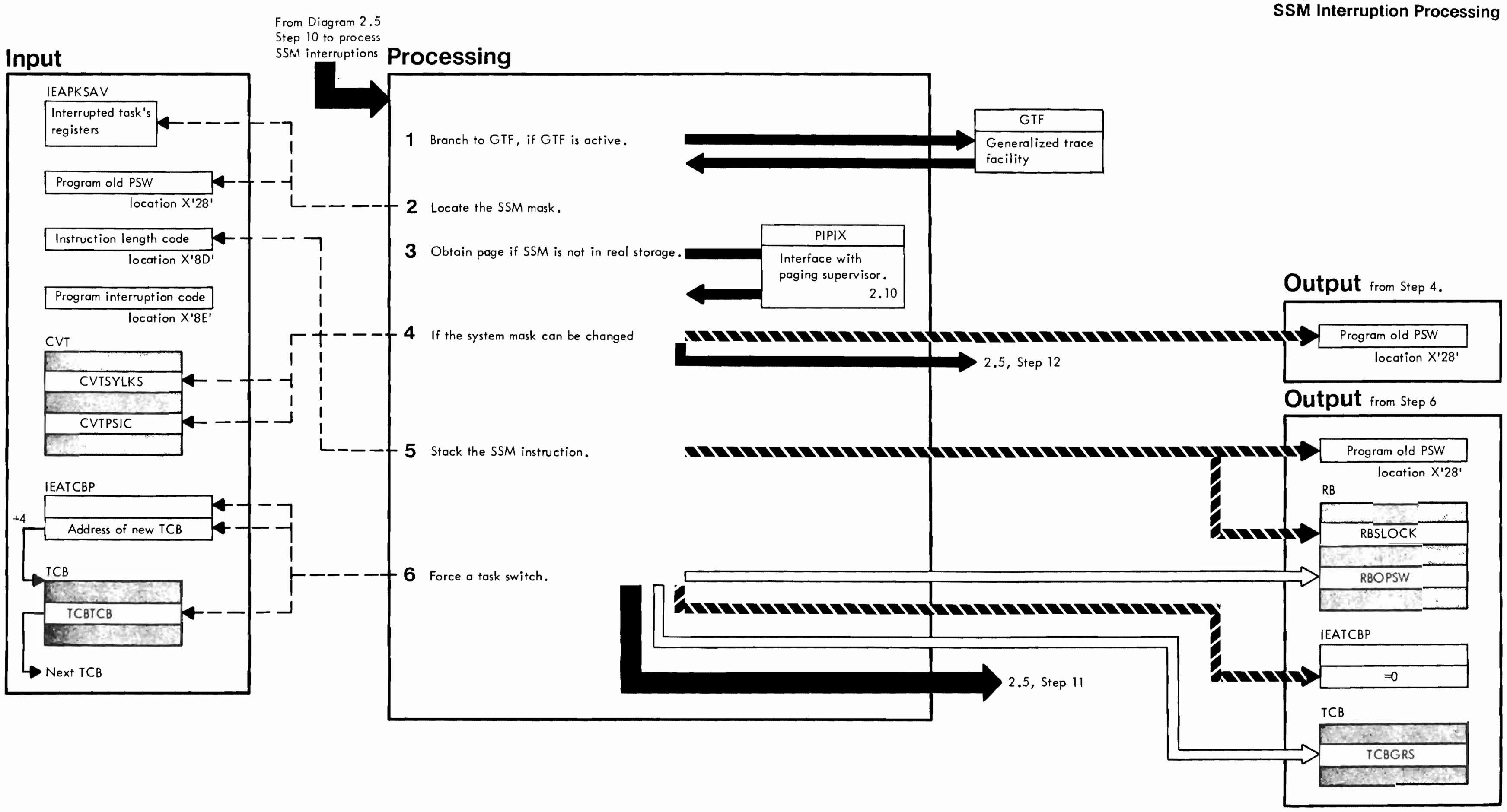
12 Return to the interrupted task.

• Diagram 2.5 Program Interruption Handling (Module IEAVNV00)

NOTES	RCUTINE NAME	LABEL
1 The program new PSW contains the address of IEAQPK00 and is set to supervisor state, protection key 0, with I/O and external interruptions disabled, and dynamic address translation inhibited. The general registers are the same as when the interruption occurred.		
2 The Program Interruption Handler (Program FLIH) is entered at IEAQPK00 whenever the CPU detects any of the hexadecimal interruption codes 0-18, 40, and 80. The processing performed is determined by these codes. The first function performed is saving the register contents in a special save area in lower real storage IEAPKSAV (no base register is needed, so no register contents are destroyed).	IEAQPK00	
3 The Program FLIH ensures that the CVT pointer at X'10' is correct.	IEAQPK00	
4 The Program FLIH branches to the paging supervisor error routine with register 0, bits 14 and 15, set to zero if the error is minor (program old PSW ≠ key 0) or with register 0, bits 14 and 15, set to zero if the error is major (indicating that the operating system was in control at the time of the failure). For both major and minor errors, register 1, bytes 3 and 4 = X'0001' to indicate entry from the Program FLIH. For minor errors, Program FLIH enables translation exceptions, and the paging supervisor tries to recover. The Program FLIH regains control only if the error is minor. For major errors, the paging supervisor places the system in a disabled wait state.	IEAQPK00 IEAPSER	 PISPEC

NOTES	ROUTINE NAME	LABEL
6 The Program FLIH branches to the GTF routine if the program interruption code is X'40' (indicating a Monitor Call interruption) or if the CVTSTATE flag is on (indicating GTF is in process for a previous Monitor Call). GTF returns to the Program FLIH only if the recursion cannot be handled. Normally, GTF returns directly to the interrupted routine or to the dispatcher.	IEAQPK00	PIGTF
7 If CVTSEIC = 1, a second exit routine is in control. There can be no valid program interruptions other than MC interruptions in second exit routines. The Program FLIH recursion flag tested is in IEAVPIRF. The ABTERM Prologue 1 routine is used to terminate the current task.	IEAQPK00 IEAQAB01	PIOAB
8 The Program FLIH restores registers 0, 1, and 15 and branches to CVTTRACE.	IEAQPK00 IEAQTRCE	 TRPI
9 If PVTPGMCK=0 the paging supervisor was in control when the program interruption occurred. PFLIH exits to the PAGEHOOK routine of the paging supervisor which puts the system in a disabled wait state.		
10 Program interruption codes: X'0011'=page fault; X'0013' =SSM; X'0010'=segment translation exception (changed to X'0004'=protection violation program check).		
11 A task switch is indicated when TCBABTRM is set or when the current TCB equals the new TCB (IEATCBP=IEATCBP+4). The Program FLIH saves registers in the TCB and the old PSW in the RB.	IEAQPK00	PIDPCH2
12 The appropriate segment table address is placed in control register 1, general registers are restored, and control is returned directly to the interrupted routine via LPSW of the program old PSW.	IEAQPK00	PIRTRN1

• Diagram 2.6
SSM Interruption Processing



1 Branch to GTF, if GTF is active.

2 Locate the SSM mask.

3 Obtain page if SSM is not in real storage.

4 If the system mask can be changed

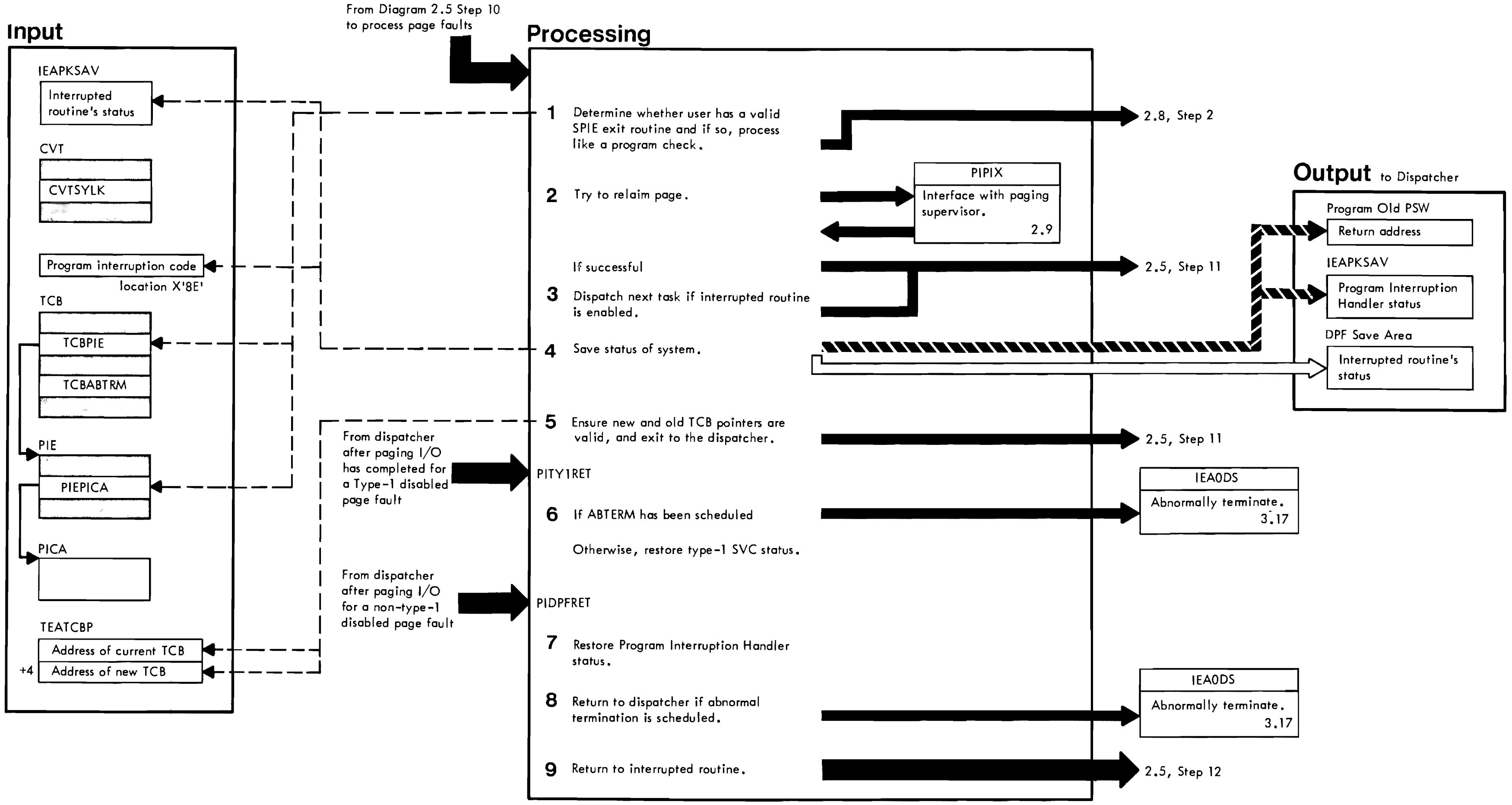
5 Stack the SSM instruction.

6 Force a task switch.

Diagram 2.6 SSM Interruption Processing (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
1 A branch instruction is issued for the SSM class via a HOOK macro instruction. If event monitoring for that class is not in effect, the hooking routine returns.	IEAQPK00	PICKSSM
2 The Program FLIH obtains the address of the instruction that caused the interruption by backing up the next sequential instruction address (NSI) portion of the old PSW by the instruction length (ILC). It then adds the base (bits 0-3 of byte 2) and displacement specified in the instruction. (If the base register is zero, the base value is zero. Otherwise, the register contents are obtained from IEAPKSAV.) The Program FLIH then tests the operation code to determine whether the interruption was caused by direct issuance of the SSM instruction, or by the EXECUTE instruction. If this is an EXECUTE instruction, the base and displacement sum is the address of the SSM instruction, and the above procedure is repeated to obtain the mask. When the SSM instruction is found, the Program FLIH adds the index register value to the base/displacement sum to obtain the address of the mask.	IEAQPK00	PIFLSSM0
3 A Load Real Address instruction is used to determine whether the SSM operand is in real storage. If the mask is not in real storage, the Program FLIH creates a pseudo-page fault and branches to the paging supervisor program interruption extension (IEAPIX). If the return code from IEAPIX is 0 (indicating that paging I/O is required), the program old PSW is reset to be reexecuted after the paging operation is completed. The request is treated as a page fault. However, if the page is reclaimed, processing continues at Step 4.	IEAQPK00 IEAPIX	PIFLSSM1 PIPIX1
4 The old PSW system mask is changed if it is not already equal to the SSM operand and if the supervisor lock is off (or if the paging task is the current task). If these conditions are in effect, bits 6 and 7 of the SSM operand are used to replace bits 6 and 7 of the user's old PSW. These bits regulate I/O and external interruptions. Otherwise, processing continues at Step 5.	IEAQPK00	PISMKTST
5 The RB is set nondispatchable (RBSLOCK=1) and the program old PSW NSI is backed up by the length in the ILC so that it points to the instruction that caused the interruption.	IEAQPK00	PIPAGSUP
6 The recursion flag is set, the new TCB address is set to zero to force a task switch, the program old PSW is stored in the RBOPSW field, the register contents in IEAPKSAV are moved into the current TCB (TCBGRS), and control is passed to the dispatcher.	IEAQPK00	PIDPCH2

• Diagram 2.7
Page Fault Processing

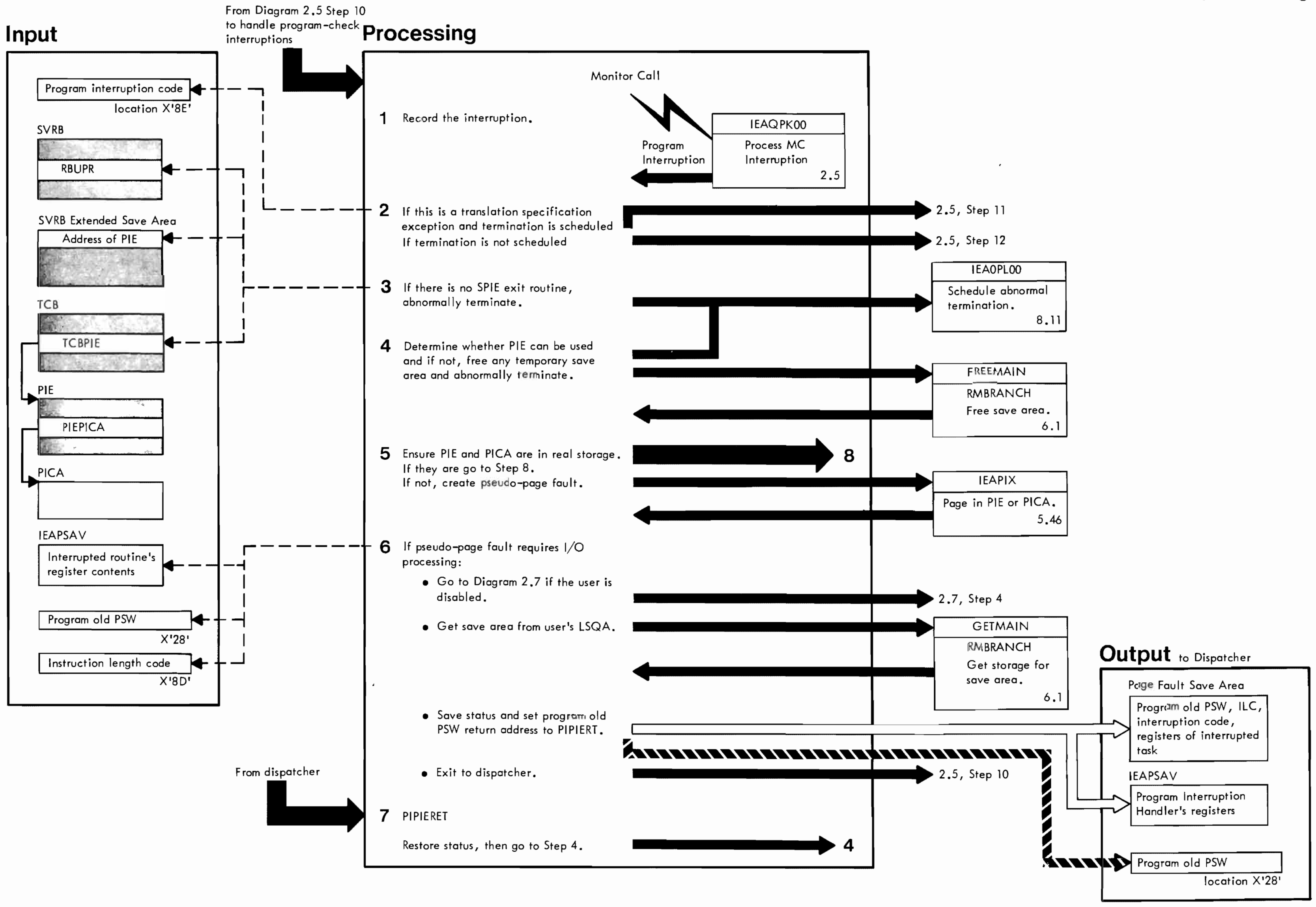


Input
IEAPKSAV
Interrupted routine's status
CVT
CVTSYLK
Program interruption code
location X'8E'
TCB
TCBPIE
TCBABTRM
PIE
PIEPICA
PICA
TEATCBP
Address of current TCB
+4
Address of new TCB
From Diagram 2.5 Step 10 to process page faults
Processing
1 Determine whether user has a valid SPIE exit routine and if so, process like a program check.
2.8, Step 2
2 Try to relaim page.
PIPIX
Interface with paging supervisor.
2.9
If successful
2.5, Step 11
3 Dispatch next task if interrupted routine is enabled.
4 Save status of system.
5 Ensure new and old TCB pointers are valid, and exit to the dispatcher.
2.5, Step 11
From dispatcher after paging I/O has completed for a Type-1 disabled page fault
PITY1RET
6 If ABTERM has been scheduled
IEA0DS
Abnormally terminate.
3.17
Otherwise, restore type-1 SVC status.
From dispatcher after paging I/O for a non-type-1 disabled page fault
PIDPFRET
7 Restore Program Interruption Handler status.
8 Return to dispatcher if abnormal termination is scheduled.
IEA0DS
Abnormally terminate.
3.17
9 Return to interrupted routine.
2.5, Step 12
Output to Dispatcher
Program Old PSW
Return address
IEAPKSAV
Program Interruption Handler status
DPF Save Area
Interrupted routine's status

Diagram 2.7 Page Fault Processing (Module IEAVNV00)

NOTES		ROUTINE NAME	LABEL
1	If the program old PSW is in problem program state and the extended PICA flag (TCBPIE17) is set in the TCB, the Program FLIH ensures that the TCB points to a valid PIE in real storage which points to a valid PICA also in real storage. The PIE must not be in use for another program interruption and the PICA must indicate that the exit routine can process the page fault.		
2		IEAQPK00	PIPIX
3	After it has been determined that a page fault needs I/O processing, if the user is enabled, exit can be made directly to the dispatcher since the task switch and status information are valid. However, if the user is disabled, it is necessary to save the status of the system at the time of the interruption.	IEAQPK00	PIPFIO
4	The Program FLIH saves the program old PSW, ILC, and registers at the time of the interruption in a special save area (DPF save area). Then, it saves its own register contents in IEAPKSAV, sets the program old PSW to return to PIDPFRET, and, if it is a type-1 SVC routine that was interrupted, saves SVC registers, the SVC old PSW, ILC, and sets the program old PSW to return to PITY1RET.	IEAQPK00	PIDFXIT
5		IEAQPK00	PITSKS2
6	ABTERM has been scheduled if TCBABTRM = 1.	IEAQPK00	PITY1RET
7		IEAQPK00	PIDPFRET

• **Diagram 2.8 (Steps 1-7)**
Program-Check Interruption Handling



1 Record the interruption.

2 If this is a translation specification exception and termination is scheduled
If termination is not scheduled

3 If there is no SPIE exit routine, abnormally terminate.

4 Determine whether PIE can be used and if not, free any temporary save area and abnormally terminate.

5 Ensure PIE and PICA are in real storage.
If they are go to Step 8.
If not, create pseudo-page fault.

6 If pseudo-page fault requires I/O processing:
- Go to Diagram 2.7 if the user is disabled.
- Get save area from user's LSQA.
- Save status and set program old PSW return address to PIPIERT.
- Exit to dispatcher.

7 PIPIERET
Restore status, then go to Step 4.

Diagram 2.8 (Steps 1-7) Program-Check Interruption Handling (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
1	IEAQPK00	PIPROGTF
2 If the program interruption code is X'12' (translation exception), no further processing is necessary and the Program FLIH exits to the dispatcher for a task switch or returns to the interrupted routine via LPSW.	IEAQPK00	PIDPCH
3 If the interrupted routine is in problem program state, it is necessary to determine whether the user has specified a SPIE exit routine to handle this interruption. The TCBPIE field indicates this. If the interrupted routine is in supervisor state, a PIE may exist for an SVC routine. If so, field RBUPR is on and the PIE address is in the SVRB's extended save area. If there is no SPIE, the Program FLIH resets the Program FLIH recursion flag, and exits to the ABTERM Prologue routine to schedule abnormal termination of the interrupted routine.	IEAQPK00	PLOGADR
6 When paging I/O is required for a PIE or PICA and the user is enabled, a test is made to determine whether there is a dynamic save area. If there is not, RMBRANCH is used to acquire the save area from the user's LSQA. Following this, or if there already was a save area, status is saved as follows: the program old PSW, ILC, interruption code, and the registers at the time of the interruption are saved, and the program old PSW in low storage is set to reenter the Program FLIH at PIPIERET in key 0, supervisor state, and disabled. The Program FLIH save area in low storage (IEAPSAV) is then set with the current contents of Program FLIH's registers, the interrupted routine's registers and PSW are moved to the TCB and RB respectively, and control is passed to the dispatcher.	IEAQPK00 PIPIX2	PICHKPIE PIPGPICA PIPIPEIO

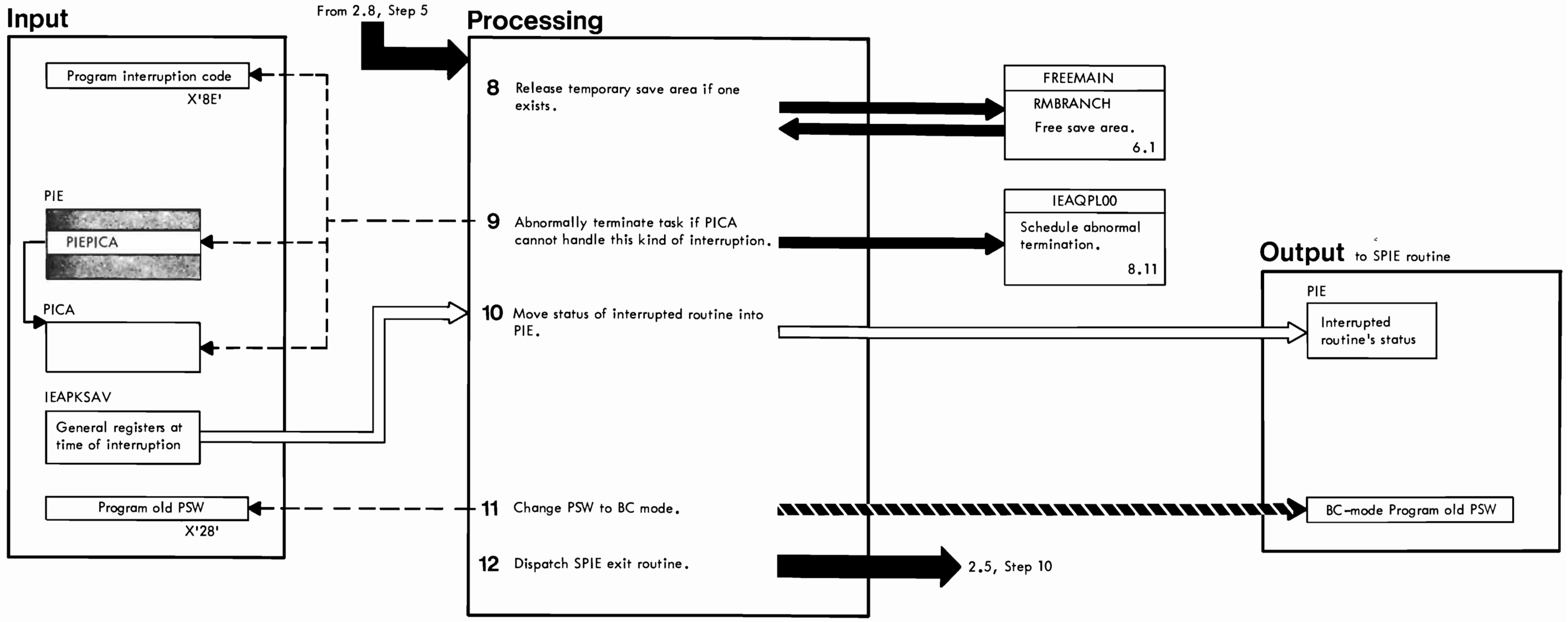

Input
From 2.8, Step 5
Processing
Program interruption code
X'8E'
PIE
PIEPICA
PICA
IEAPKSAV
General registers at time of interruption
Program old PSW
X'28'
8 Release temporary save area if one exists.
9 Abnormally terminate task if PICA cannot handle this kind of interruption.
10 Move status of interrupted routine into PIE.
11 Change PSW to BC mode.
12 Dispatch SPIE exit routine.
FREEMAIN
RMBRANCH
Free save area.
6.1
IEAQPL00
Schedule abnormal termination.
8.11
2.5, Step 10
Output to SPIE routine
PIE
Interrupted routine's status
BC-mode Program old PSW

Diagram 2.8 (Steps 8-12) Program-Check Interruption Handling (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
9 Each interruption code has corresponding bits in PICAITMK which indicate whether the interruption is to be processed by the user's exit routine.	IEAQPK00	PICHECK
10 When it has been determined that the interruption is to be handled by the user's exit routine, the PIE is marked "in use," registers 14-2 are moved from IEAPKSAV to the PIE, the program old PSW is reformatted from EC mode to BC mode and saved in the PIE, the address of a special SVC 3 (EXIT) instruction is placed in the register 14 slot of the Program FLIH save area (this is used by Exit to determine if the issuer is a SPIE exit routine), and the address of the exit routine is placed in the register 15 slot and in the program old PSW. Following this, the program old PSW mask is set from the PICA program mask, and, if this is for a page fault, the address of the page causing the interruption is saved in the register 0 slot of the Program FLIH save area.		

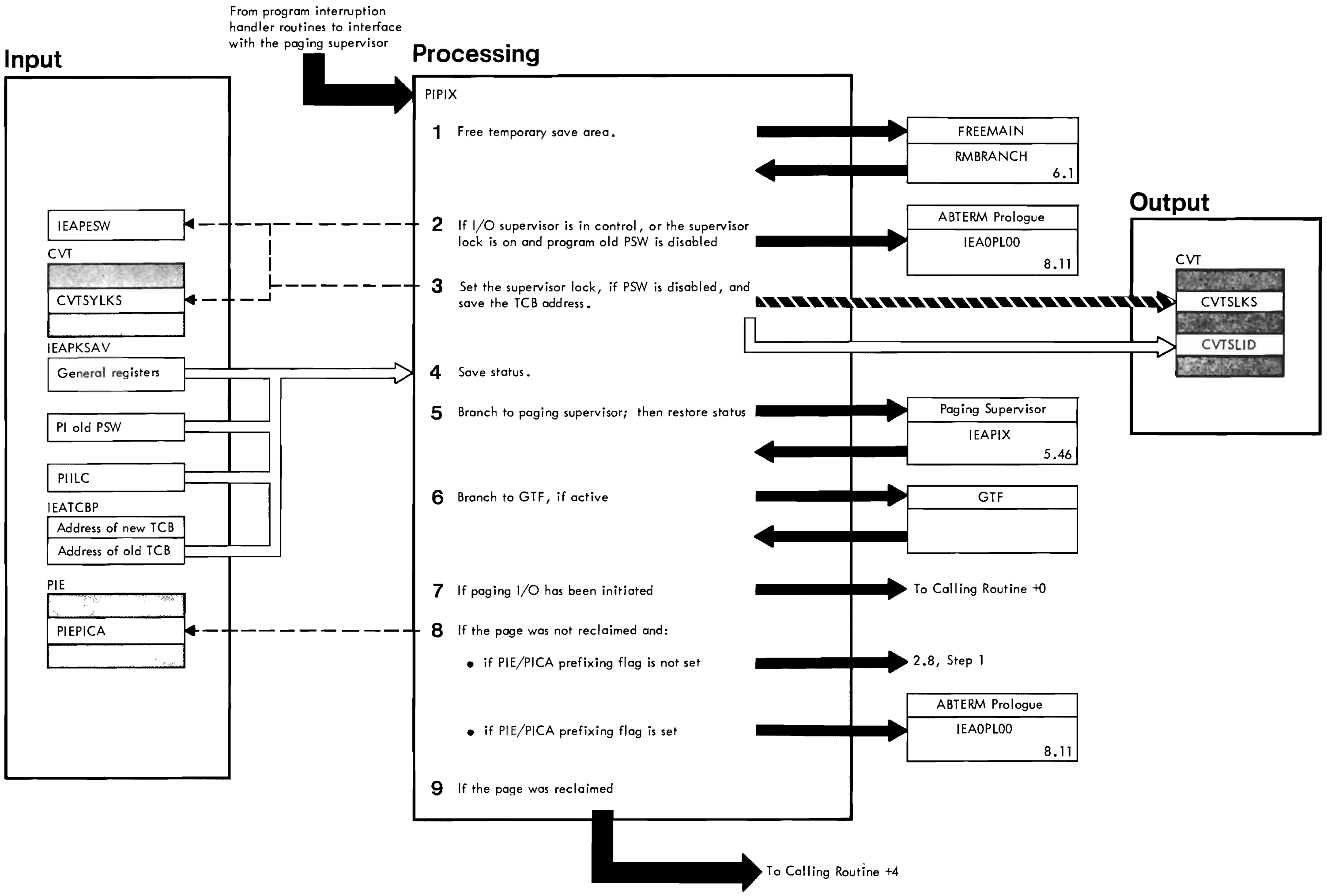
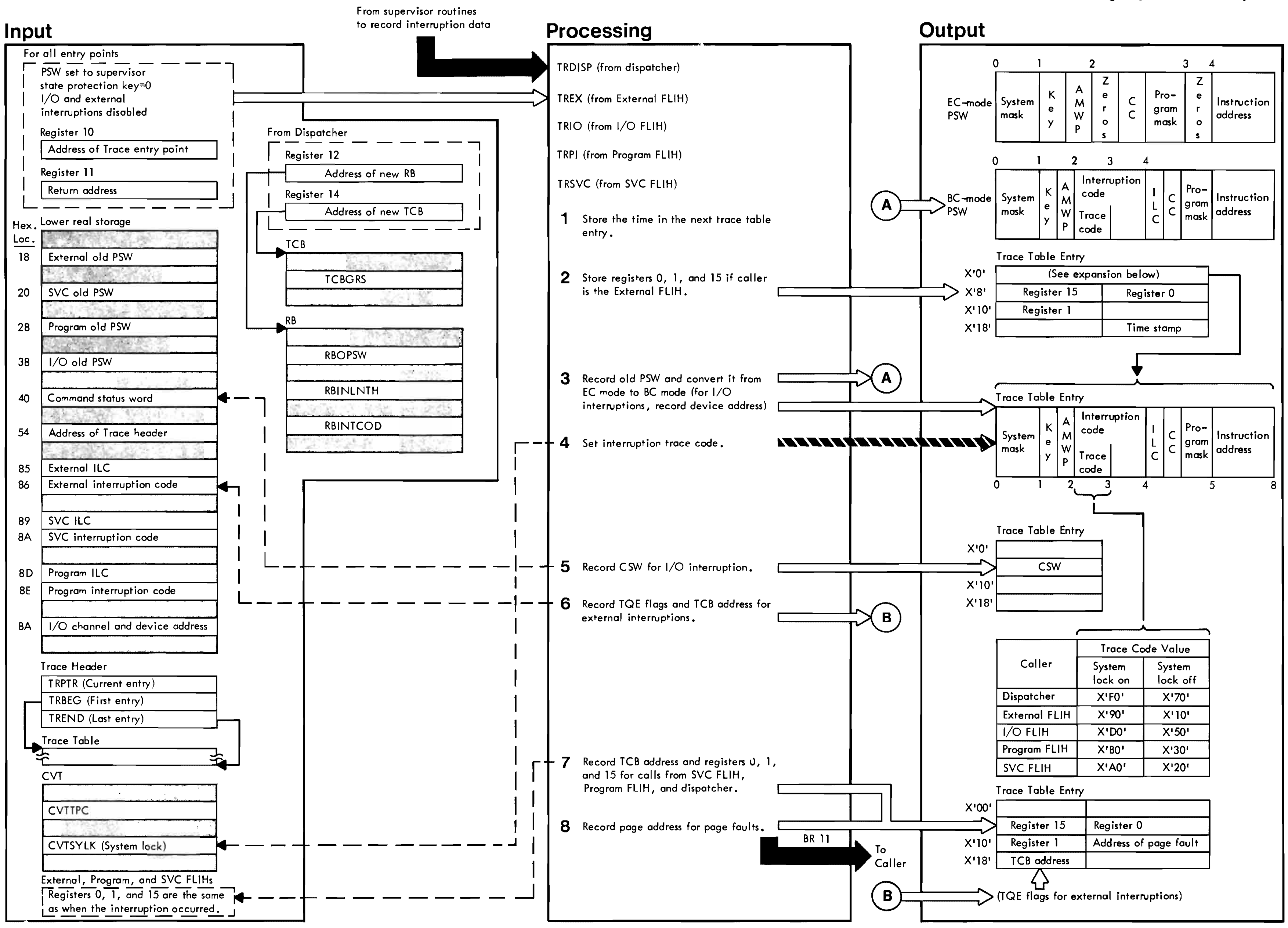

From program interruption handler routines to interface with the paging supervisor
Input
IEAPESW
CVT
CVTSYLKS
IEAPKSAV
General registers
PI old PSW
PIILC
IEATCBP
Address of new TCB
Address of old TCB
PIE
PIEPICA
Processing
PIPIX
1 Free temporary save area.
2 If I/O supervisor is in control, or the supervisor lock is on and program old PSW is disabled
3 Set the supervisor lock, if PSW is disabled, and save the TCB address.
4 Save status.
5 Branch to paging supervisor; then restore status
6 Branch to GTF, if active
7 If paging I/O has been initiated
8 If the page was not reclaimed and:
• if PIE/PICA prefixing flag is not set
• if PIE/PICA prefixing flag is set
9 If the page was reclaimed
FREEMAIN
RMBRANCH
6.1
ABTERM Prologue
IEAOPL00
8.11
Paging Supervisor
IEAPIX
5.46
GTF
To Calling Routine +0
2.8, Step 1
ABTERM Prologue
IEAOPL00
8.11
To Calling Routine +4
Output
CVT
CVTSLKS
CVTSLID

Diagram 2.9 PIPIX Routine (Module IEAVNV00)

NOTES	ROUTINE NAME	LABEL
2 Before exiting to the ABTERM Prologue routine, the Program FLIH turns off the Program FLIH recursion flag.	PIPIX	
3		PIPIXDSB
4 The status must be saved because the Program Interruption Handler Extension of the paging supervisor may cause a Monitor Call program interruption.		PICALPIX
6 If the return code from the paging supervisor is 0, PIPIX resets the "PIE/PICA prefixing flag" and, if the supervisor lock has been set for this request, PIPIS sets the "inhibit second exits flag."		PINEEDIO
8 If the requester owns the supervisor lock, PIPIX sets it.		
9 If the return code from the paging supervisor is 4, the page was reclaimed. PIPIX sets the "PIE/PICA prefixing flag" and returns control to the caller+4.		PICRD4

Diagram 2.10
Tracing Supervisor Interruptions



1 Store the time in the next trace table entry.

2 Store registers 0, 1, and 15 if caller is the External FLIH.

3 Record old PSW and convert it from EC mode to BC mode (for I/O interruptions, record device address)

4 Set interruption trace code.

5 Record CSW for I/O interruption.

6 Record TQE flags and TCB address for external interruptions.

7 Record TCB address and registers 0, 1, and 15 for calls from SVC FLIH, Program FLIH, and dispatcher.

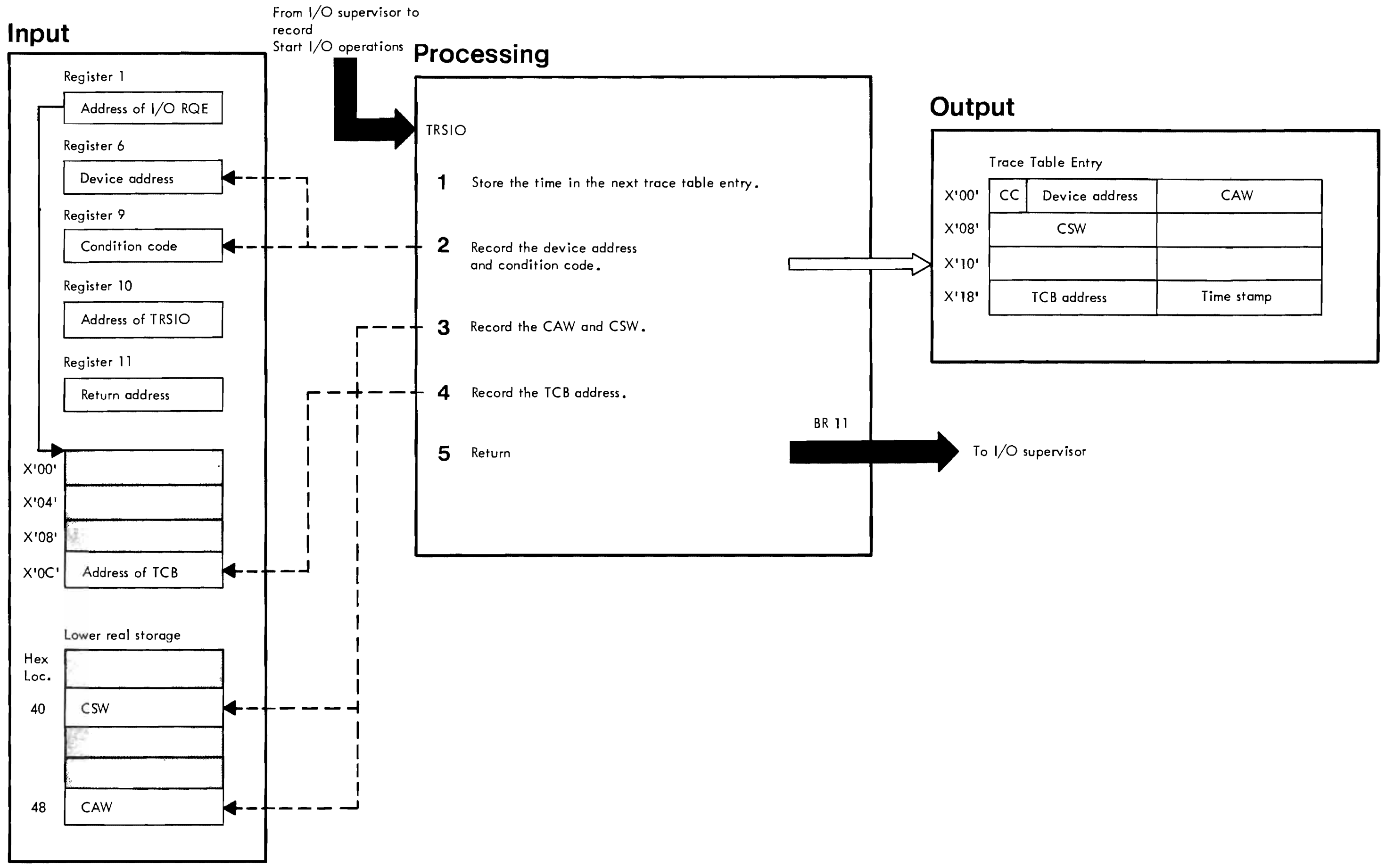
8 Record page address for page faults.

Caller	Trace Code Value	
	System lock on	System lock off
Dispatcher	X'F0'	X'70'
External FLIH	X'90'	X'10'
I/O FLIH	X'D0'	X'50'
Program FLIH	X'B0'	X'30'
SVC FLIH	X'A0'	X'20'

Diagram 2.10 Tracing Supervisor Interruptions (Module IEAQTRCE)

NOTES	ROUTINE NAME	LABEL
1 The address of the next trace table entry address is equal to TRPTR+32 if that value is less than TREND (the final entry). If TRPTR+32 equals or exceeds TREND, TRPTR is changed to equal TRBEG (the beginning address). Bits 2-5 of the time-of-day value (time stamp) are copied into the last four bits of the next trace table entry.	PRECOM	
2	TREX	
3 The ECTOBC subroutine is used to convert the PSW from EC mode to BC mode for all but I/O interruptions. For I/O interruptions, TRIO converts the I/O old PSW and replaces the instruction address field with the I/O channel and device address.	ECTOBC TRIO	
4 The trace code is set by either TRDISP, TREX, TRIO, TRPI, or TRSVC, whichever was called to record the particular event.		
5	TRIO	IO2
6 If the external interruption code is X'1004', the clock comparator caused the interruption. If the code is X'1005', the CPU timer caused the interruption. The flags of the appropriate TQE are recorded in the trace table entry.	TREX	EXT3
7 The TRDISP subroutine stores registers 0, 1, and 15 contained in the new TCB for calls from the dispatcher. For SVC and program interruptions, the current TCB and current registers 0, 1, and 15 are recorded.	TRDISP TRSVC	DSP2 TRCOMM
8 The address of the page fault or segment translation exception is stored in the trace table entry.		POSTCOM

Diagram 2.11
Tracing Start I/O Operations



SECTION 3

Task Supervision

3

Task supervision controls the execution of tasks and allocates CPU time to competing tasks. Diagram 3.1 shows the routines that make up task supervision.

Task supervision routines that are type-1 and type-2 SVC routines provide services that can be requested by problem programs or system programs by issuing a macro instruction (for example, ATTACH). The routines that are not shown as type-1 or type-2 SVCs are entered directly by other system routines.

A task is described by a TCB (task control block). The TCB contains information needed to control the execution of the task (see the description of the TCB in Section 12).

There are two types of tasks: permanent tasks and dynamic tasks. Each task is represented by one TCB with chaining fields that form two TCB queues: the task queue, which includes all active tasks in the system, and the subtask queue for the job step. (These queues are described under "The Two TCB Queues" later in this section.

Permanent Tasks

When the operating system is generated, the TCBs for the permanent system tasks are built into the nucleus. These TCBs are the beginning of the task queue (see Figure 3-1).

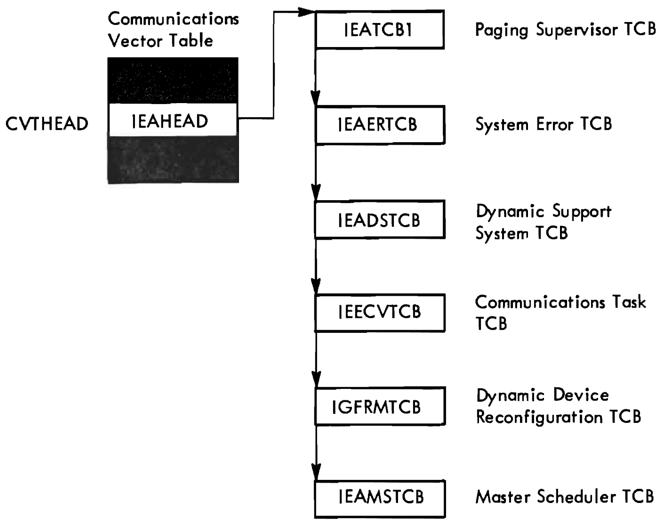
Dynamic Tasks

Dynamic tasks are created by task supervision in response to a system or user ATTACH request. When the ATTACH routine creates a task, it builds the TCB that represents the task usually in the LSQA (local system queue area) of the requester. But if the LSQA operand is specified in the ATTACH request, the ATTACH routine places the TCB and other control blocks associated with the task in a new LSQA for the new subtask.

CREATING A TASK

The following is the basic sequence of task creation: The master scheduler, which is a permanent task, attaches an initiator/terminator task, which is a job management task. Initiator/terminator tasks attach job-step tasks, which in turn attach subtasks.

The advantage of subtasking is that subtasks compete for the use of the CPU when



Note: The TCBs are queued in descending order according to dispatching priority.

Figure 3-1. Permanent TCBs that form the beginning of the task queue.

either the job-step task or one of its subtasks must wait. When a job-step task that uses subtasking must wait, it is not necessarily another job step that gains control.

THE TWO TCB QUEUES

The ATTACH routine establishes pointers in a new TCB in such a way that the TCB becomes part of two TCB queues: the task queue, and the subtask queue for a job step.

The task queue contains all TCBs in the system. On this queue, the TCBs are chained in a descending order of priority, with the permanent system tasks having the highest positions on the queue. The CHAP routine manipulates this queue when it changes the dispatching priorities of tasks, and the dispatcher tests the TCBs on this queue to determine which task should be dispatched next.

There is a subtask queue for each job step in the system. The subtask queue starts with the job-step TCB and contains a TCB for each task attached by the job-step task or one of its subtasks. The TCBs on this queue show the order in which the TCBs for the job step were created. The termination routines use the subtask queue to determine the sequence for freeing the job step's resources when the job step is abnormally terminated.

TCB Pointers

The relationship of tasks on the task queue is indicated by two chaining fields:

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

The relationship of subtasks on a subtask queue is indicated by four chaining fields:

- 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 3-2 shows a subtask queue. In the figure, task 0 is a job-step task and has two subtasks, tasks 1 and 2. Task 1 was attached first. Task 2 has one subtask, task 3.

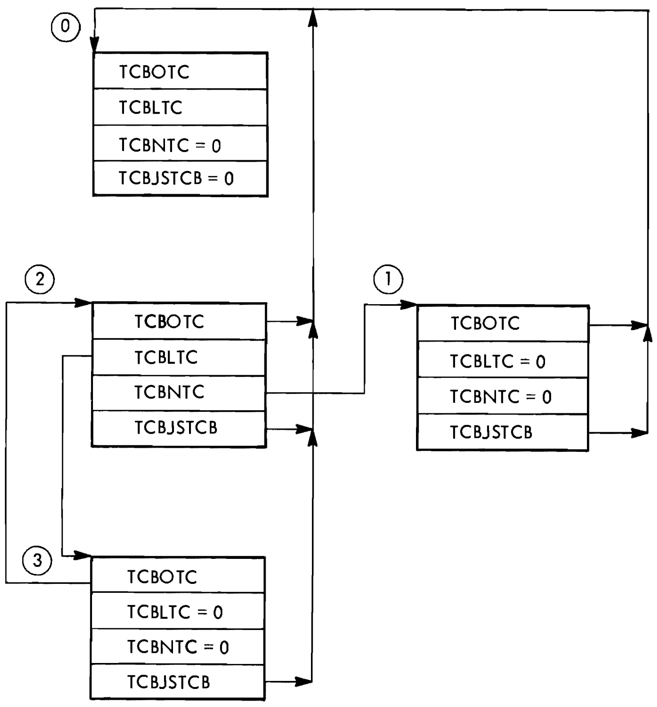


Figure 3-2. Example of a subtask queue showing how TCB pointers link a subtask to related subtasks.

ALLOCATING CPU TIME TO COMPETING TASKS

The dispatcher determines which task is to be dispatched next and passes control to the current routine of that task. The task to be dispatched next is one of the following:

- The current task, whose performance is being resumed.
- Another ready task of higher priority than the current task.
- Another ready task of lower priority than the current task, if the current task is waiting or is nondispatchable.
- Another task in the same time-sliced group.
- Another task in the APG.

The interrupted routine of the current task is given control if no supervisor routine has indicated the need for a task switch. If a task switch has been indicated, however, the dispatcher gives control to the current routine of the highest priority ready task. This task may be of higher or lower priority than the current task. The address of the "new" task's TCB is found either in the NEW TCB pointer (IEATCBP), or through a search of the task queue.

If the dispatcher does not find a routine that can be dispatched, it dispatches a dummy task which is part of the nucleus. The dummy task has no associated routines and places the CPU in an enabled wait state. After a future interruption, one of the nonready tasks may be readied by an interruption handler, and CPU execution can continue.

In addition, the dispatcher handles task and job-step timing. One or more priority levels may have been identified as time-sliced groups. For tasks within these groups, the dispatcher determines which time-sliced task is to receive control next and dispatches the task with the time interval identified for that group.

A single priority level may be identified at system generation or system initialization time as an APG (automatic priority group). (A time-slicing group cannot be identified as an automatic priority group.) Tasks within an automatic priority group are dispatched in a way that makes best use of the system's CPU and I/O resources.

The APG tasks constitute a subset of the entire task queue. In addition, the APG is divided into two subgroups as shown in Figure 3-3.

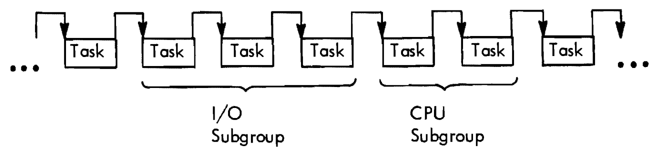


Figure 3-3. Portion of the task queue showing two subgroups in an automatic priority group.

Note that the VS2 dispatcher operates with a priority-ordered task queue. A search of the task queue proceeds from left to right. Tasks that the CHAP or ATTACH service routines place in the APG are marked as I/O-oriented tasks and are queued at the head of the APG section of the queue. This permits prompt identification of the characteristics of new APG tasks. The Task Switch routine signals task switches between APG tasks only when the current task is a CPU-oriented task and the contending task is an I/O-oriented task. This eliminates unnecessary task switching overhead between APG tasks that have essentially the same characteristics.

Tasks are dispatched and shifted within the APG according to their characteristics. Tasks are considered to be CPU-oriented if they utilize their entire time interval without voluntarily relinquishing control of the CPU. (A task may be CPU-oriented even if it uses more I/O time than CPU time.) Tasks are I/O-oriented if they seem to involve more use of I/O (paging I/O is excluded from this determination).

All tasks in the APG are dispatched with a timed interval. The decision on whether a task is I/O-oriented is based on whether the task uses its entire interval of allotted time. Unused portions of the interval are moved when page faults stop a task or a task is preempted by higher priority tasks.

Important features of the APG group are:

- Tasks within the I/O subgroup are arranged in such a way that those using smaller portions of their interval are ranked higher in the queue.

- CPU tasks receive control in a cyclic manner, thus ensuring that any available CPU time is distributed equitably among them. This ensures that throughput time for CPU-oriented tasks is in proportion to their stand-alone run time. In addition, potential I/O tasks are not kept at the bottom of the CPU subgroup indefinitely.

The algorithm that governs the selection and dispatching of APG tasks is based on the information in Figure 3-4. The following are the self-adjusting characteristics and parameters associated with the VS2 dispatcher:

1. The original time interval that is assigned to APG tasks.

2. An incremental time that can be added or subtracted from the original time interval.

3. A lower limit on the adjusted time interval.

4. An upper limit on the adjusted time interval.

5. A predetermined value expressing the desired ratio of X to Y, where X = total number of times APG tasks used their full time interval, and Y = sum of X and total number of times APG tasks voluntarily surrendered CPU control. (Note: X and Y are reset to 0 at the end of each statistics interval.)

6. A statistics interval used to determine the frequency with which the dispatching algorithm adjusts itself.

Parameters 2 through 6 can be specified at system generation or system initialization. The initial value of parameter 1 is:

(upper limit + lower limit) - 2

Throughout a statistics interval, all APG tasks are dispatched with the same time interval. Counts are kept of the X and Y values. When the statistics interval concludes, the ratio of X to Y is computed and compared to the value of parameter 5. If the calculated value is lower, the resolution of the algorithm is not adequate to detect enough CPU-oriented tasks. Accordingly, the current value of parameter 1 is decreased by the magnitude of parameter 2. Conversely, if too few I/O-oriented tasks are being identified by the algorithm, the value of parameter 1 is increased.

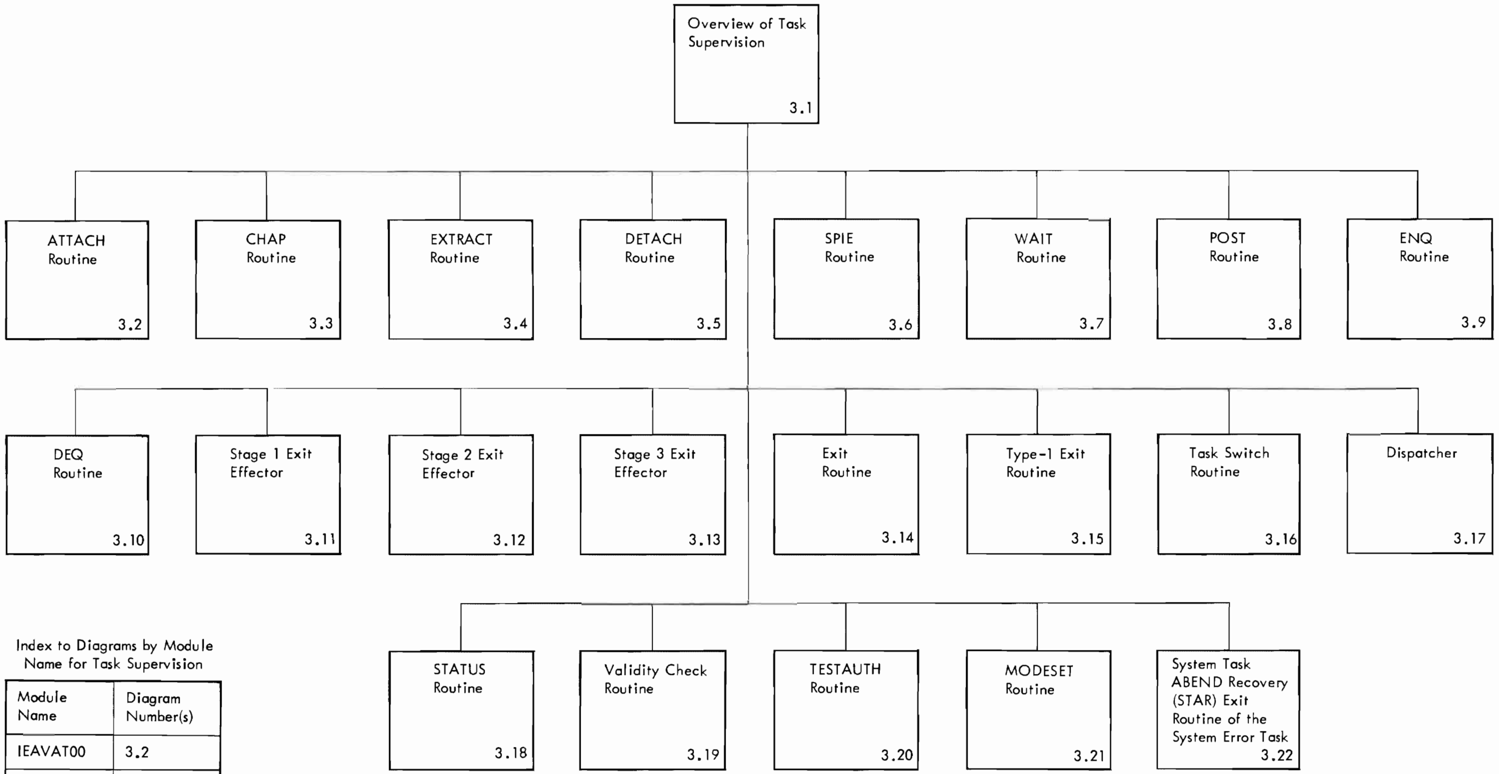
SUMMARY OF TASK SUPERVISION

In summary, the ATTACH routine of task supervision creates TCBs for all but permanent system tasks. After it queues these TCBs, other task supervision routines maintain the status of all tasks in the system. A short summary of each routine accompanies the diagrams, which follow.

Reason for Loss of CPU Control	New Task Status	Action Taken
Original Task Status - I/O		
Preemption by task above the APG	I/O	Save unused portion of interval; pass control to preempting task.
Time interval end	CPU	Set up full interval for next dispatch of task; mark task CPU-oriented and queue at top of CPU subgroup; locate next task by searching from top of APG.
Voluntary surrender	I/O	Set up full interval for next dispatch of task; queue task within I/O subgroup based on portion of interval used; locate next task by searching from top of APG.
Original Task Status - CPU		
Preemption by I/O task or by task above the APG.	CPU	Save unused portion of interval; pass control to preempting task.
Time interval end	CPU	Set up full interval for next dispatch of task; queue task at end of CPU subgroup; locate next task by searching from top of CPU subgroup.
Voluntary surrender	I/O	Set up full interval for next dispatch of task; queue task at end of I/O subgroup and mark task I/O-oriented; locate next task by searching from top of CPU subgroup.

Figure 3-4. Factors in the dispatching of APG tasks.

• **Diagram 3.0**
Task Supervision
Visual Table of Contents



Index to Diagrams by Module Name for Task Supervision

Module Name	Diagram Number(s)
IEAVAT00	3.2
IEAVCH00	3.3
IEAVED02	3.5
IEAVEF00	3.11
IEAVENQ1	3.9, 3.10
IEAVET00	3.14
IEAVMODE	3.21
IEAVNU00	3.12, 3.13, 3.15--3.17, 3.19, 3.20, 3.22
IEAVSETS	3.18
IEAVSY50	3.7, 3.8
IEAVTB00	3.4, 3.6

• **Diagram 3.1**
Overview of Task Supervision

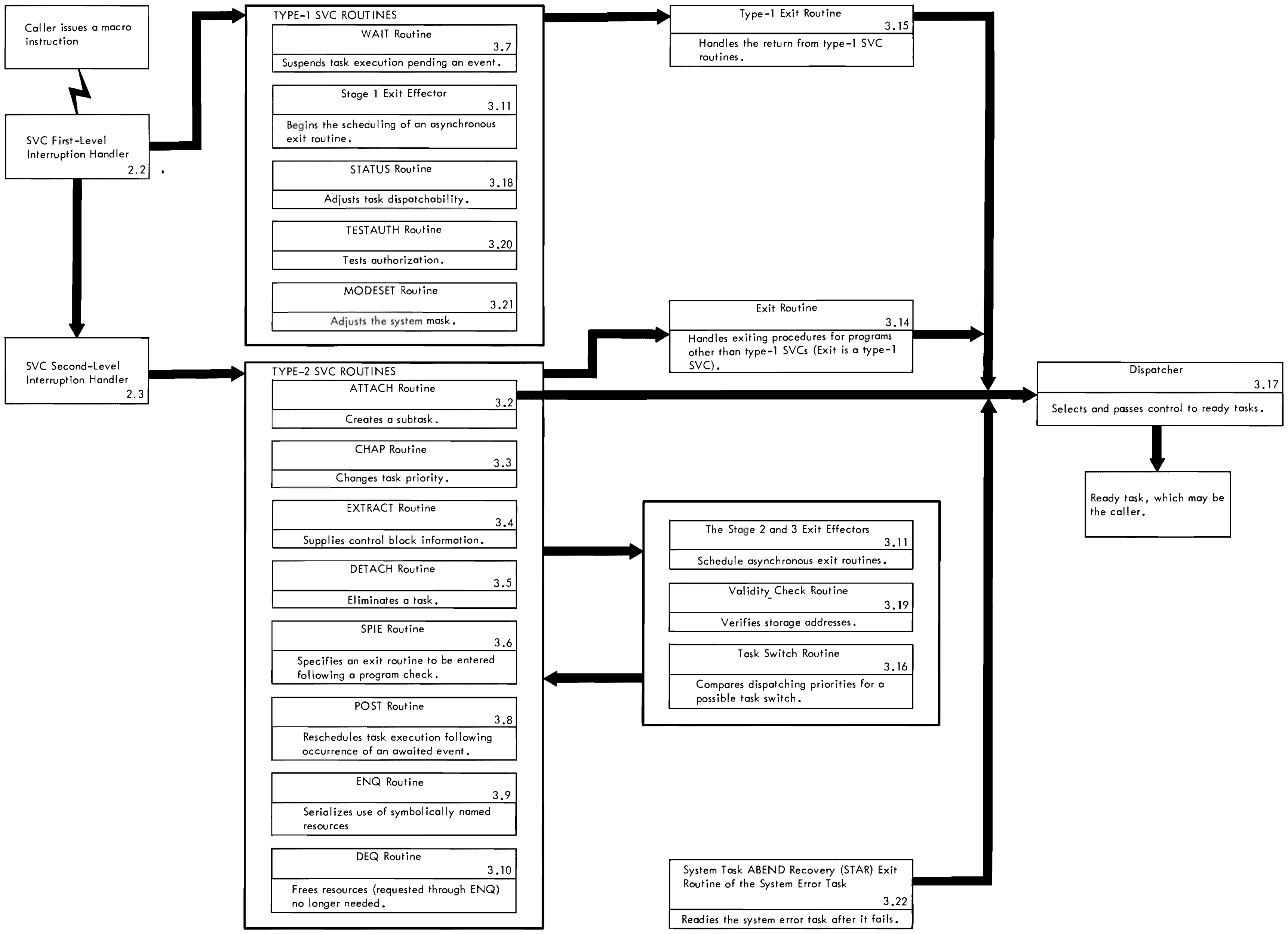
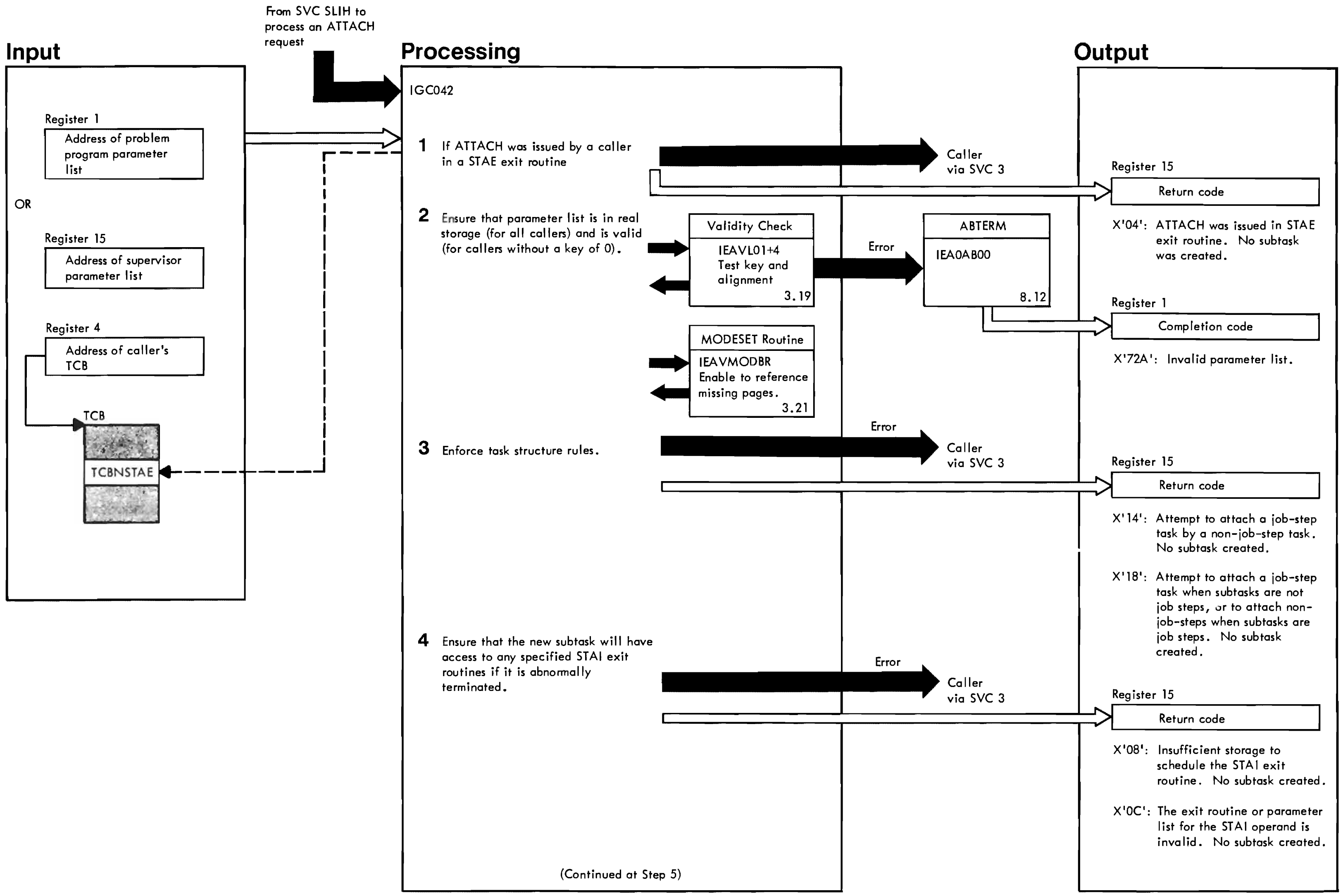

Caller issues a macro instruction
SVC First-Level Interruption Handler 2.2
SVC Second-Level Interruption Handler 2.3
TYPE-1 SVC ROUTINES
WAIT Routine 3.7
Suspends task execution pending an event.
Stage 1 Exit Effector 3.11
Begins the scheduling of an asynchronous exit routine.
STATUS Routine 3.18
Adjusts task dispatchability.
TESTAUTH Routine 3.20
Tests authorization.
MODESET Routine 3.21
Adjusts the system mask.
TYPE-2 SVC ROUTINES
ATTACH Routine 3.2
Creates a subtask.
CHAP Routine 3.3
Changes task priority.
EXTRACT Routine 3.4
Supplies control block information.
DETACH Routine 3.5
Eliminates a task.
SPIE Routine 3.6
Specifies an exit routine to be entered following a program check.
POST Routine 3.8
Reschedules task execution following occurrence of an awaited event.
ENQ Routine 3.9
Serializes use of symbolically named resources
DEQ Routine 3.10
Frees resources (requested through ENQ) no longer needed.
Type-1 Exit Routine 3.15
Handles the return from type-1 SVC routines.
Exit Routine 3.14
Handles exiting procedures for programs other than type-1 SVCs (Exit is a type-1 SVC).
The Stage 2 and 3 Exit Effectors 3.11
Schedule asynchronous exit routines.
Validity Check Routine 3.19
Verifies storage addresses.
Task Switch Routine 3.16
Compares dispatching priorities for a possible task switch.
System Task ABEND Recovery (STAR) Exit Routine of the System Error Task 3.22
Readies the system error task after it fails.
Dispatcher 3.17
Selects and passes control to ready tasks.
Ready task, which may be the caller.

Diagram 3.2 (Steps 1-4)
ATTACH Routine



From SVC SLIH to process an ATTACH request

Input

Register 1
Address of problem program parameter list

OR

Register 15
Address of supervisor parameter list

Register 4
Address of caller's TCB

TCB
TCBNSTAE

Processing

IGC042

1 If ATTACH was issued by a caller in a STAE exit routine

Caller via SVC 3

2 Ensure that parameter list is in real storage (for all callers) and is valid (for callers without a key of 0).

Validity Check
IEAVL01+4
Test key and alignment
3.19

Error

ABTERM
IEA0AB00
8.12

MODESET Routine
IEAVMODBR
Enable to reference missing pages.
3.21

3 Enforce task structure rules.

Error

Caller via SVC 3

4 Ensure that the new subtask will have access to any specified STAI exit routines if it is abnormally terminated.

Error

Caller via SVC 3

(Continued at Step 5)

Output

Register 15
Return code

X'04': ATTACH was issued in STAE exit routine. No subtask was created.

Register 1
Completion code

X'72A': Invalid parameter list.

Register 15
Return code

X'14': Attempt to attach a job-step task by a non-job-step task. No subtask created.

X'18': Attempt to attach a job-step task when subtasks are not job steps, or to attach non-job-steps when subtasks are job steps. No subtask created.

Register 15
Return code

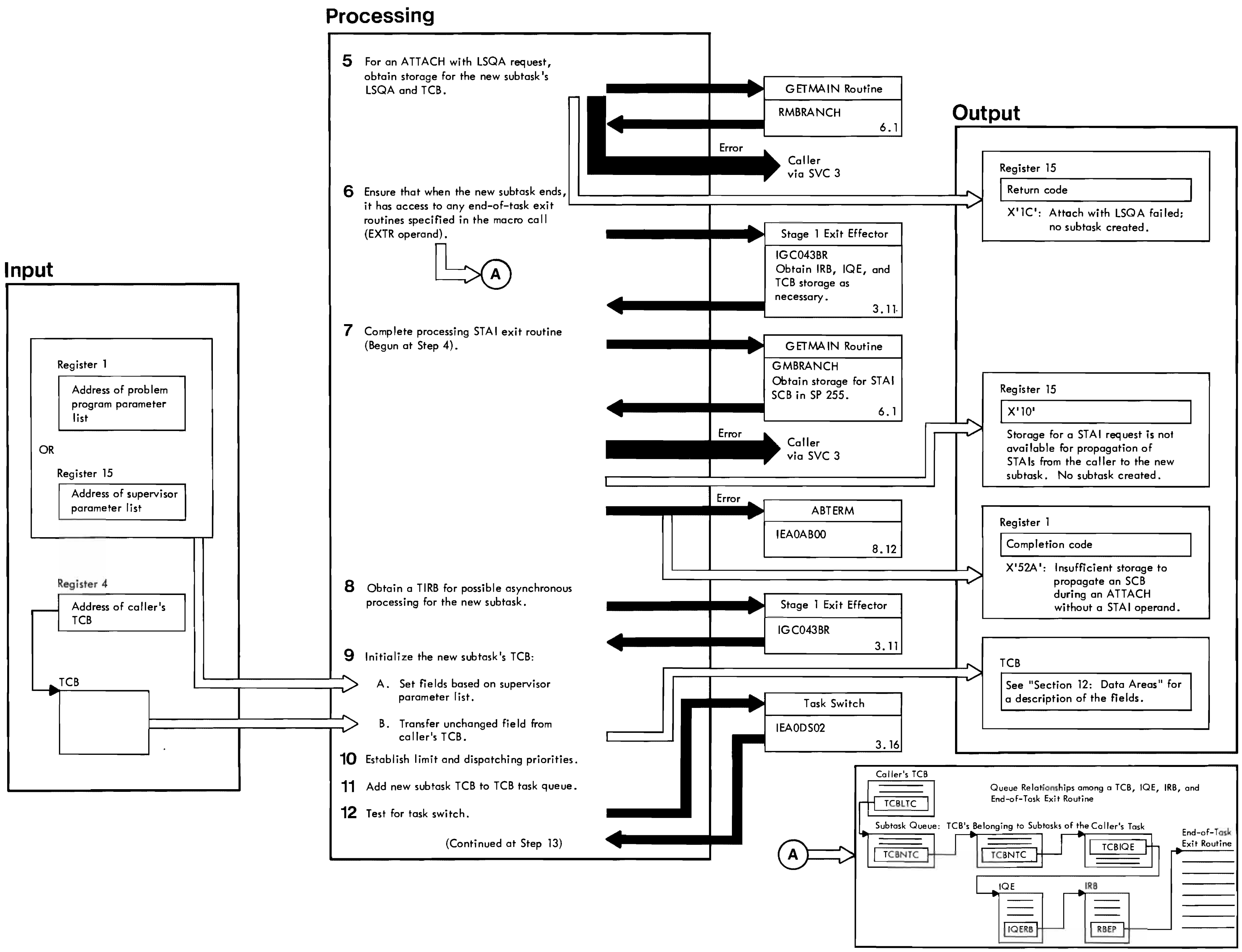
X'08': Insufficient storage to schedule the STAI exit routine. No subtask created.

X'0C': The exit routine or parameter list for the STAI operand is invalid. No subtask created.

Diagram 3.2 (Steps 1-4) ATTACH Routine (Module IEAVAT00)

NOTES	ROUTINE NAME	LABEL
The ATTACH routine permits a problem program or system program to attach a subtask. The ATTACH routine creates a TCB to represent the subtask, places control information in the new TCB, allocates storage to the subtask, places the new TCB on the two TCB queues, and schedules linkage to contents supervision to obtain the first program to be executed for the new subtask. When the new subtask has the highest priority among the ready tasks, the specified program is given control. Caller's with a key of 0 can specify the creation of one or more LSQA (local system queue area) segments to the TCB for the new subtask. This type of request is called ATTACH with LSQA.	ATTACH	IGC042
2		NOSTAE
3		TASKRULE
4 If the ATTACH macro instruction was issued with the STAI operand, the ATTACH routine ensures that the new subtask will have access to the specified STAI exit routine if the subtask is scheduled for abnormal termination. The STAI operand has no effect on ATTACH with LSQA requests. STAI exit routines perform the same functions for a subtask that a STAE exit routine performs. ATTACH issues a STAI SVC to create a dummy SCB (STA control block) which points to the exit routine, and then queues it to the calling routine's TCB. (The new subtask's TCB does not yet exist.)		STAETEST

Diagram 3.2 (Steps 5-12)
ATTACH Routine



Processing

5 For an ATTACH with LSQA request, obtain storage for the new subtask's LSQA and TCB.

6 Ensure that when the new subtask ends, it has access to any end-of-task exit routines specified in the macro call (EXTR operand).

7 Complete processing STAI exit routine (Begun at Step 4).

8 Obtain a TIRB for possible asynchronous processing for the new subtask.

9 Initialize the new subtask's TCB:

A. Set fields based on supervisor parameter list.

B. Transfer unchanged field from caller's TCB.

10 Establish limit and dispatching priorities.

11 Add new subtask TCB to TCB task queue.

12 Test for task switch.

(Continued at Step 13)

Output

X'1C': Attach with LSQA failed; no subtask created.

Storage for a STAI request is not available for propagation of STAIs from the caller to the new subtask. No subtask created.

X'52A': Insufficient storage to propagate an SCB during an ATTACH without a STAI operand.

See "Section 12: Data Areas" for a description of the fields.

Diagram 3.2 (Steps 5-12) ATTACH Routine (Module IEAVAT00)

NOTES	ROUTINE NAME	LABEL
5	ATTACH	KEYOTEST
6 At ETXR, ATTACH determines whether the requester has supplied the address of an asynchronous (end-of-task) exit routine. If so, ATTACH must get storage for and initialize an IRB and an IQE. • An IRB (interruption request block) is used in scheduling an asynchronous exit routine. • An IQE (interruption queue element) represents a pending request for the asynchronous exit routine. An IRB with the specified address may already exist because another task has requested the same asynchronous exit routine. In this case, ATTACH calls GETMAIN to get storage for an IQE, and uses the existing IRB. If no IRB exists, ATTACH calls the Stage 1 Exit Effector (CIRB) to build the IRB.		ETXR

Type of ATTACH	Status of IRB	IQE	TCB
With LSQA	Exists	RMBRANCH	RMBRANCH
With LSQA	None exists CIRB(IGC043BR)	CIRB (IGC043BR)	RMBRANCH
Without LSQA	Exists	RMBRANCH	RMBRANCH
Without LSQA	None exists CIRB	CIRB	CIRB

NOTES	ROUTINE NAME	LABEL
7 ATTACH propagates to the new subtask's TCB. The STAI SCBs that were queued to the calling routine's TCB.		PROPSTAI
8		COMN
10		ATOK2
12 If this is not an ATTACH with LSQA, a dummy RB is presented to the Task Switch routine. This RB is in a ready state.		NEWTEST1

Processing

13 Prepare the ATTACH SVRB to be used as the RB for the new subtask.

14 If an ECB for the new subtask has been specified, check its validity.

Validity Check
IEAOVL01
Test key and alignment.
3.19

ABTERM
IEA0AB00
8.12

15 Resolve storage and module ownership for the new subtask.

ABTERM
IEA0AB00
8.12

16 Pass control to the LINK routine.

LINK Routine (4.2)
Step 2

Output

Register 1

Completion codes

X'42A':	An invalid ECB address was supplied.
X'12A':	An attempt was made to give a shared or owned subpool to the subtask.
X'22A':	An attempt was made to give or share a supervisor subpool.
X'32A':	An attempt was made to give the job pack area while it contained CDEs whose use count was not 0.

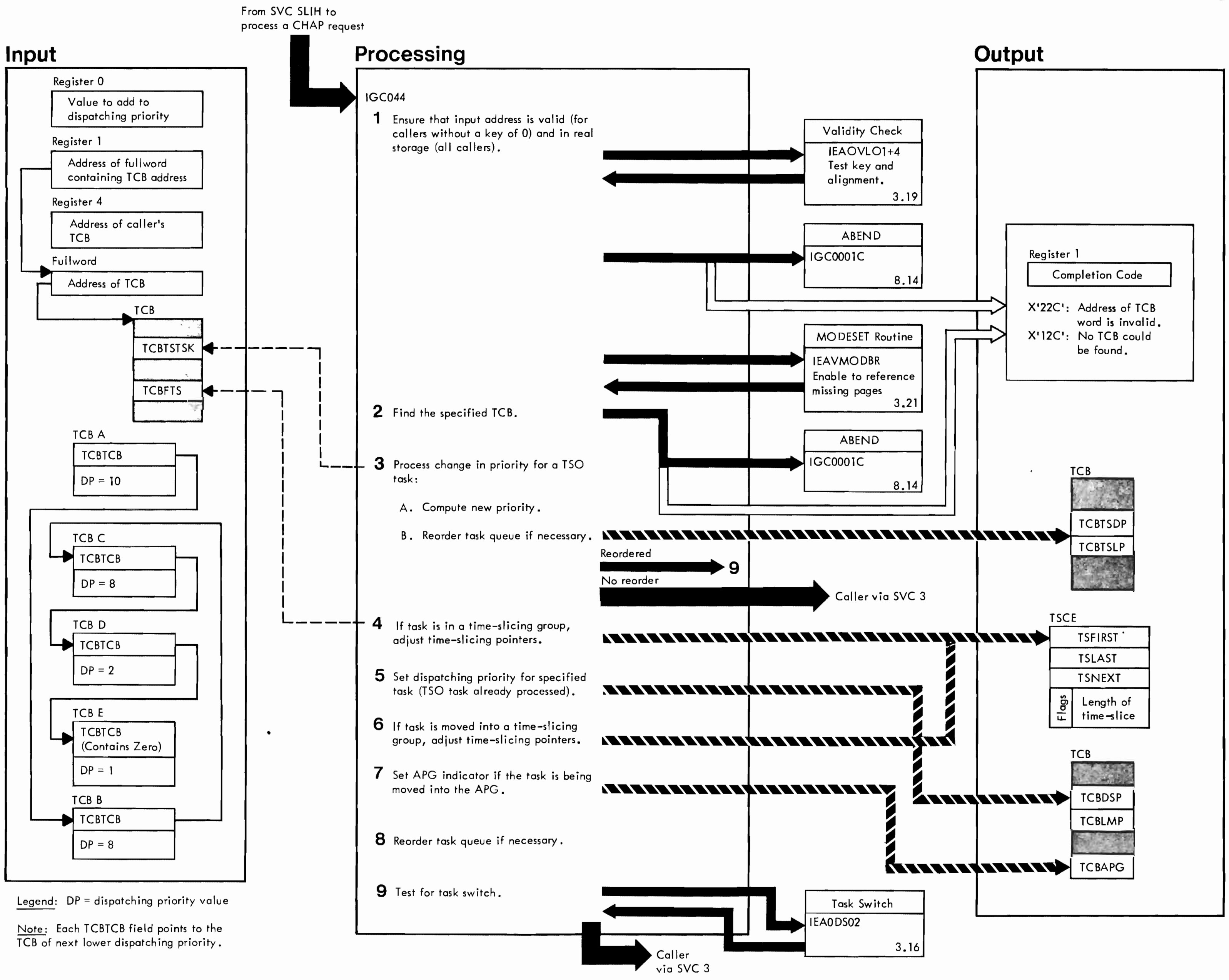
Register 0
Address of EP name or PDS DE

Register 1
Address of DCB

Register 4
Address of new TCB

Diagram 3.2 (Steps 13-16) ATTACH Routine (Module IEAVAT00)

NOTES	ROUTINE NAME	LABEL
13 If this is an ATTACH with LSQA, ATTACH copies the requester's SVRB into the new subtask's LSQA, and frees the requester's SVRB.	ATTACH	
14		ECB
15 The ATTACH routine allocates subpools to the new subtask according to the input parameters. The GIVE parameter causes allccation of subpools to the subtask for its exclusive use. The SHARE parameter permits the programs of the originating task and the programs of the new subtask to share the same subpools. ATTACH exits to the dispatcher, which allows any possible task switch to take place. When the new subtask is dispatched, processing resumes at Step 16.		ATOK4
16 If a register save area is requested, 72 bytes are obtained from subpool 250. At NOSAVE, the registers are set up to pass control to contents supervisicn.		GETSAVE NOSAVE



IGC044

1 Ensure that input address is valid (for callers without a key of 0) and in real storage (all callers).

2 Find the specified TCB.

3 Process change in priority for a TSO task:

A. Compute new priority.

B. Reorder task queue if necessary.

4 If task is in a time-slicing group, adjust time-slicing pointers.

5 Set dispatching priority for specified task (TSO task already processed).

6 If task is moved into a time-slicing group, adjust time-slicing pointers.

7 Set APG indicator if the task is being moved into the APG.

8 Reorder task queue if necessary.

9 Test for task switch.

Legend: DP = dispatching priority value

Note: Each TCBTCB field points to the TCB of next lower dispatching priority.

Diagram 3.3 CHAP Routine (Module IEAVCH00)

NOTES	ROUTINE NAME	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.	CHAP	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 is as follows.		
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 places in execution the current program having 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.		
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 the by SVC Second-Level Interruption Handler (SLIH), and no validity check of the address is required.		OWNTCB
2 If a valid address is supplied in register 1, 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.		KEYTEST
The CHAP routine does not make this test if the caller's TCB (address in register 4) is the subject.		
3 If the priority of a TSO task (TCBTSTSK =1) is being		DOCHAP
adjusted, only TSO tasks within the requester's subgroup, as specified by the TJBX (time sharing job block extension) can be affected by the CHAP request.		TSTASK
The TCB fields that CHAP changes are TCBTSDP, containing the dispatching priority; and TCBTSLP, containing the limit priority. The new priorities, shown below in column 2, are determined by CHAP, which adds the number input in register 0 to the current priority to obtain the new total.		

Priority Requested	New-Priorities
Total ≤ 0	TCBTSDP = 0
Total > caller's TCBTSLP	TCBTSDP, TCBTSLP = TCBTSLP of caller
Total ≤ caller's	TCBTSDP = total; if TCBTSDP ≥ TCBTSLP, set TCBTSLP = new TCBTSDP

(Label TSXREST)

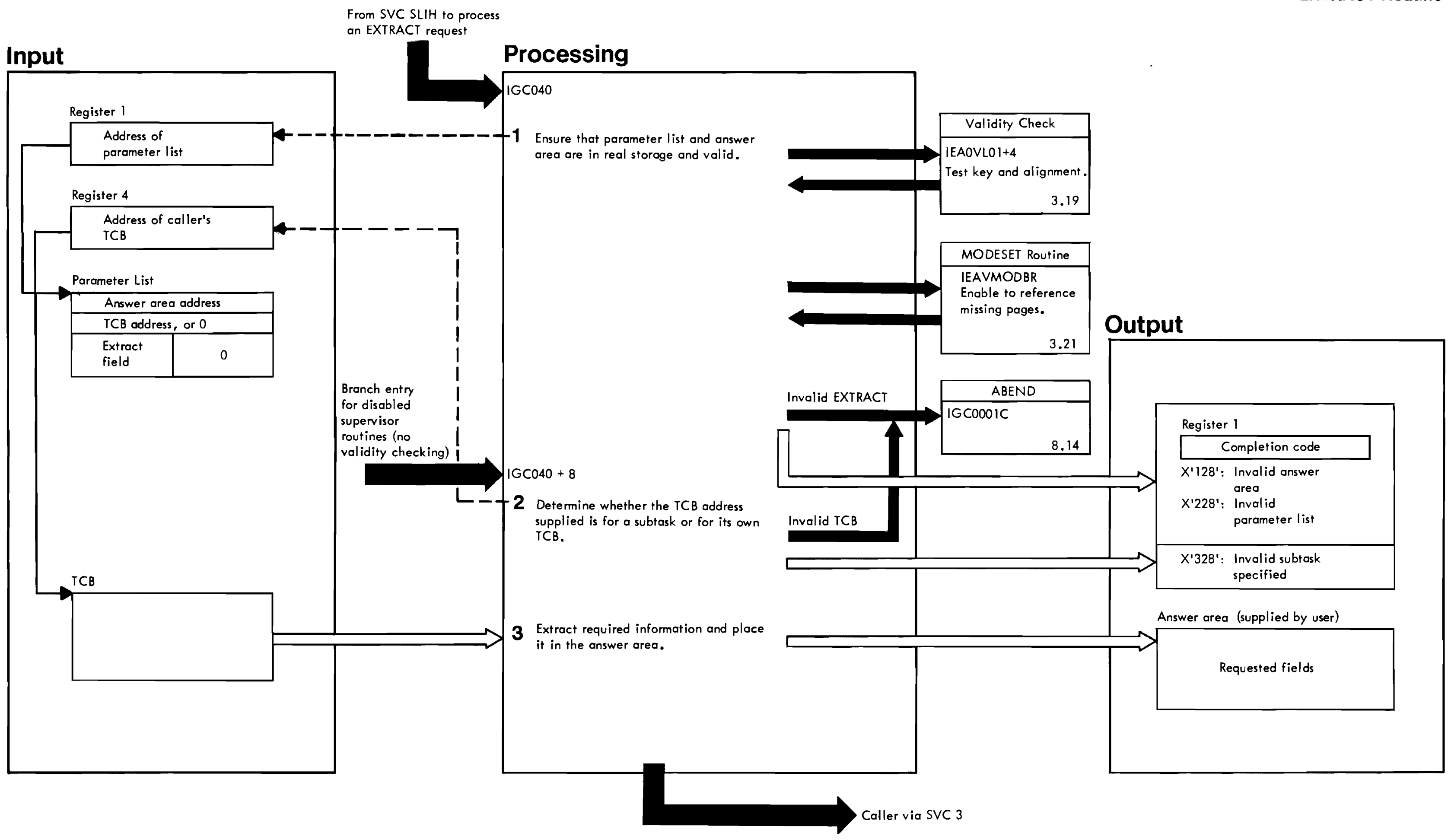
NOTES	ROUTINE NAME	LABEL
If the priority of the LOGON task is set lower than any of the subtasks, the dispatching priority of these subtasks is set at the new level of the LOGON task.		

NOTES	ROUTINE NAME	LABEL
4 If TCBFTS = 1, the subject task is a member of a time-sliced group. TCBFTS is set to 0, because a change in the dispatching priority shifts a time-sliced task out of its group.	CHAP	TSTSBIT
Pointers must be adjusted in the TSCE (time-slicing control element) to reflect the change in priority. If the address of the TCB is the same as the address in the TSFIRST, TSNEXT, and TSLAST fields in the TSCE, all three fields are set equal to zero.		CHKTSCE

Field Containing TCB Address	Meaning and CHAP Processing
TSFIRST but not TSLAST	Specified task is not the only one in the group. CHAP places address of the next lower-level task on task queue into TSFIRST.
TSLAST and TSNEXT but not TSFIRST	Specified task is last in the group and next to be dispatched. CHAP places address from TSFIRST into TSNEXT and address of next-higher-level on the TCB queue into TSLAST.
TSLAST but not TSNEXT	CHAP places address of next higher-level task TCB on task queue into TSLAST.

NOTES	ROUTINE NAME	LABEL
5 When changing the dispatching priority of a non-TSO task, CHAP follows the rules listed in the notes for step 3. For non-TSO tasks, the dispatching priority is in field TCBDSP, and the limit priority is in field TCBLMP.		DOCHAP1
6 The pointers in the TSCE are set to accommodate the new TCB, which is pointed to by TSLAST. If this is the only task in the group, TSFIRST and TSNEXT must also point to the new task.		CHK4TSP
8 The TCB is queued according to its dispatching priority, but at the end of its priority level. There are two exceptions: (1) a new APG task is marked I/O and queued at the top of its priority level, and (2) any non-TSO tasks with a priority of zero are queued ahead of TSO tasks with a priority of zero.		LOCPLACE

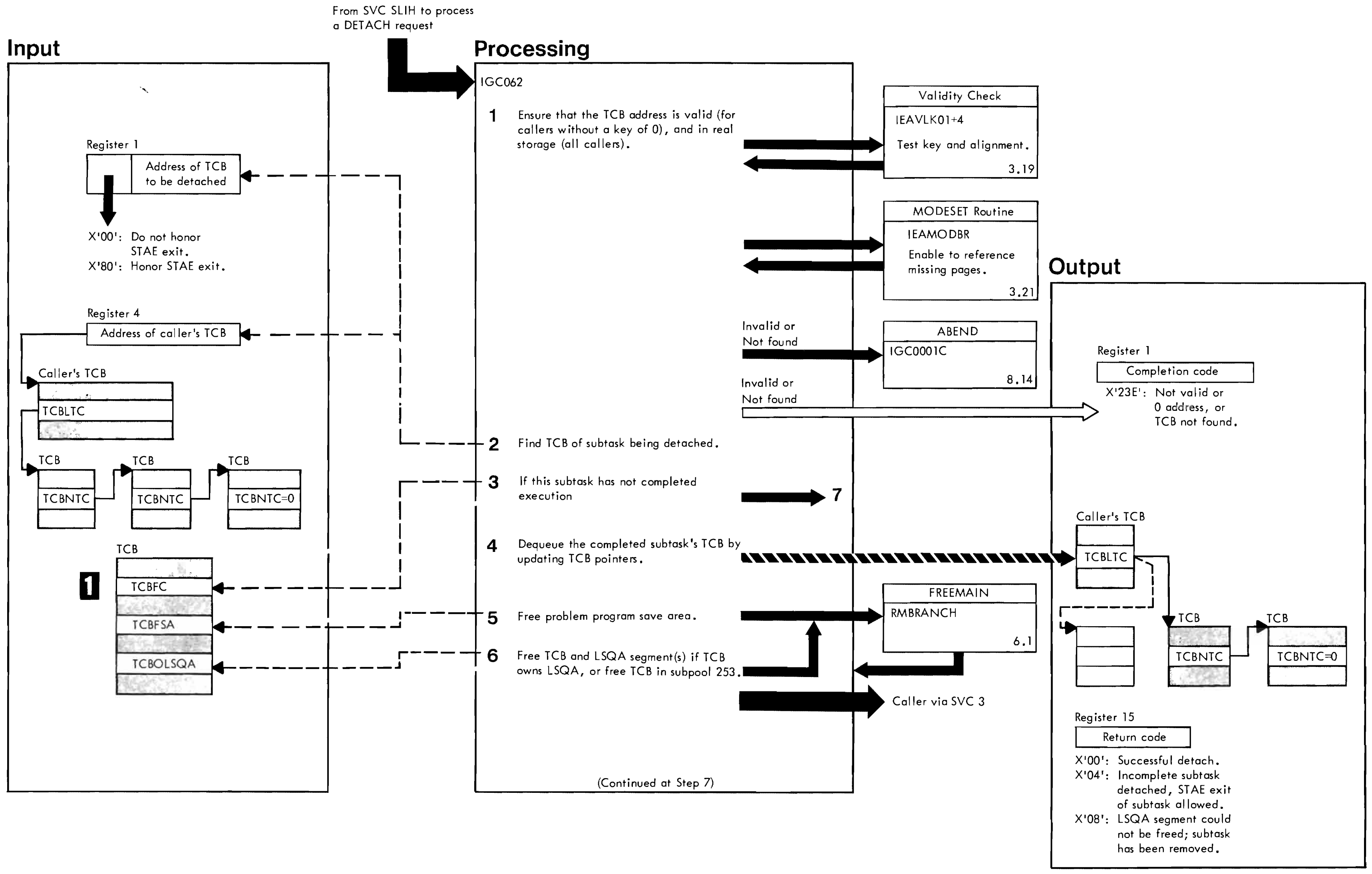
Diagram 3.4
EXTRACT Routine



Input
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
TCB
From SVC SLIH to process an EXTRACT request
Processing
IGC040
1 Ensure that parameter list and answer area are in real storage and valid.
Branch entry for disabled supervisor routines (no validity checking)
IGC040 + 8
2 Determine whether the TCB address supplied is for a subtask or for its own TCB.
3 Extract required information and place it in the answer area.
Validity Check
IEAOVL01+4
Test key and alignment.
3.19
MODESET Routine
IEAVMODBR
Enable to reference missing pages.
3.21
ABEND
IGC0001C
8.14
Invalid EXTRACT
Invalid TCB
Caller via SVC 3
Output
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 3.4 EXTRACT Routine (Module IEAVTB00)

NOTES	ROUTINE NAME	LABEL
The EXTRACT routine permits a problem program or system program to request information from its own TCB or the TCB of a subtask. Through the 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 or subsidiary control block is stored in a caller-specified list in the caller's region.	EXTRACT	IGC040
2 For non-key-0 caller's only, the input TCB address must match either the address of the caller's TCB or the address of a TCB representing one of the caller's subtasks.		GOODSTUF
If the caller does not supply a TCB address, EXTRACT uses the address of the caller's TCB, placed in register 4 by SVC SLIH. EXTRACT bypasses Step 2 if it uses the register 4 address.		
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 <u>OS/VS Supervisor Services and Macro Instructions</u> for a list of the TCB fields that can be extracted.) For each bit set, EXTRACT copies appropriate information from the TCB into the answer area.		QSTORE



From SVC SLIH to process a DETACH request

Input

Register 1: Address of TCB to be detached

X'00': Do not honor STAE exit.
X'80': Honor STAE exit.

Register 4: Address of caller's TCB

Caller's TCB: TCBLTC

TCB: TCBNTC → TCB: TCBNTC → TCB: TCBNTC=0

1 TCB: TCBFC, TCBFSA, TCBOLSQA

Processing

IGC062

1 Ensure that the TCB address is valid (for callers without a key of 0), and in real storage (all callers).

2 Find TCB of subtask being detached.

3 If this subtask has not completed execution → 7

4 Dequeue the completed subtask's TCB by updating TCB pointers.

5 Free problem program save area.

6 Free TCB and LSQA segment(s) if TCB owns LSQA, or free TCB in subpool 253.

Caller via SVC 3

(Continued at Step 7)

Validity Check: IEAVLK01+4 — Test key and alignment. 3.19

MODESET Routine: IEAMODBR — Enable to reference missing pages. 3.21

Invalid or Not found → ABEND: IGC0001C 8.14

Invalid or Not found → Output

FREEMAIN: RMBRANCH 6.1

Output

Register 1: Completion code

X'23E': Not valid or 0 address, or TCB not found.

Caller's TCB: TCBLTC

TCB: TCBNTC → TCB: TCBNTC=0

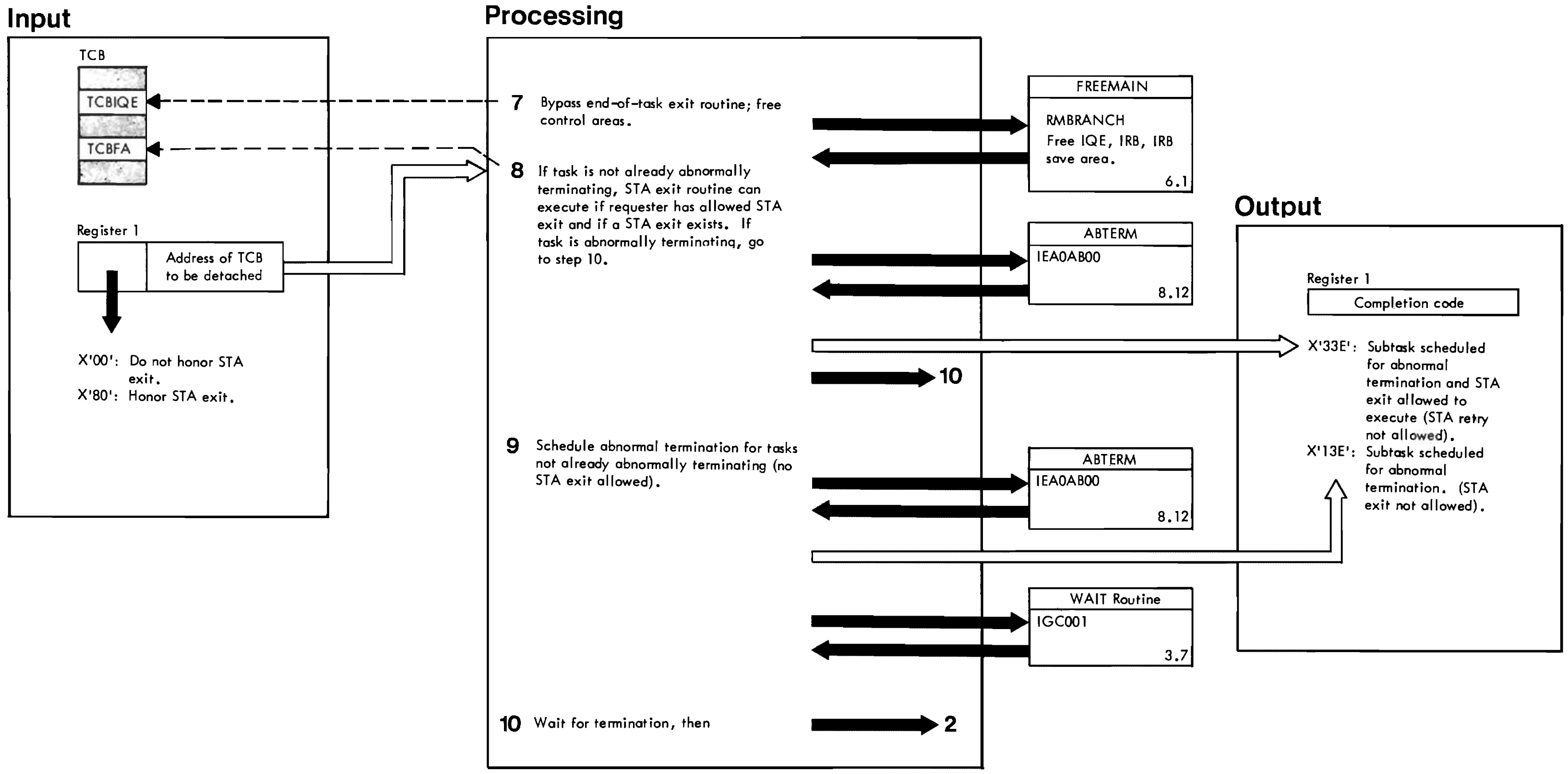
Register 15: Return code

X'00': Successful detach.
X'04': Incomplete subtask detached, STAE exit of subtask allowed.
X'08': LSQA segment could not be freed; subtask has been removed.

Diagram 3.5 (Steps 1-6) DETACH Routine (Module IEAVED02)

NOTES	ROUTINE NAME	LABEL
The DETACH routine permits a program being executed for an originating task to detach its subtask if the subtask has been normally or abnormally terminated. The DETACH routine checks whether the address of the subtask's TCB passed to the DETACH routine is valid, and whether the subtask has been terminated. It dequeues the subtask's TCB from the subtask queue and frees storage areas belonging to the subtask, including the subtask's TCB. If the caller specifies an invalid TCB address, the DETACH routine abnormally terminates the caller's task. If the subtask has not been normally or abnormally terminated before the DETACH request, DETACH abnormally terminates it as part of its processing.	DETACH	IGC062
1 Meaning of TCB fields referred to: TCBFC Equals 1 if the task has been terminated. TCBFSA Address of problem program save area. TCBOLSQA Equals 1 if the subtask owns LSQA segment(s). TCBIQE Address of an IQE or zero. TCBFA Equals 1 if the task is abnormally terminating.		
2 DETACH compares the input TCB address (in register 1) with the TCBs that represent the caller's subtasks until: • The TCB that represents the subtask to be detached is found, or • DETACH finds a 0 in the TCBNTC field, and the specified TCB has not been found.		CHKRET
5		DTFREE

Diagram 3.5 (Steps 7-10)
DETACH Routine



7 Bypass end-of-task exit routine; free control areas.

8 If task is not already abnormally terminating, STA exit routine can execute if requester has allowed STA exit and if a STA exit exists. If task is abnormally terminating, go to step 10.

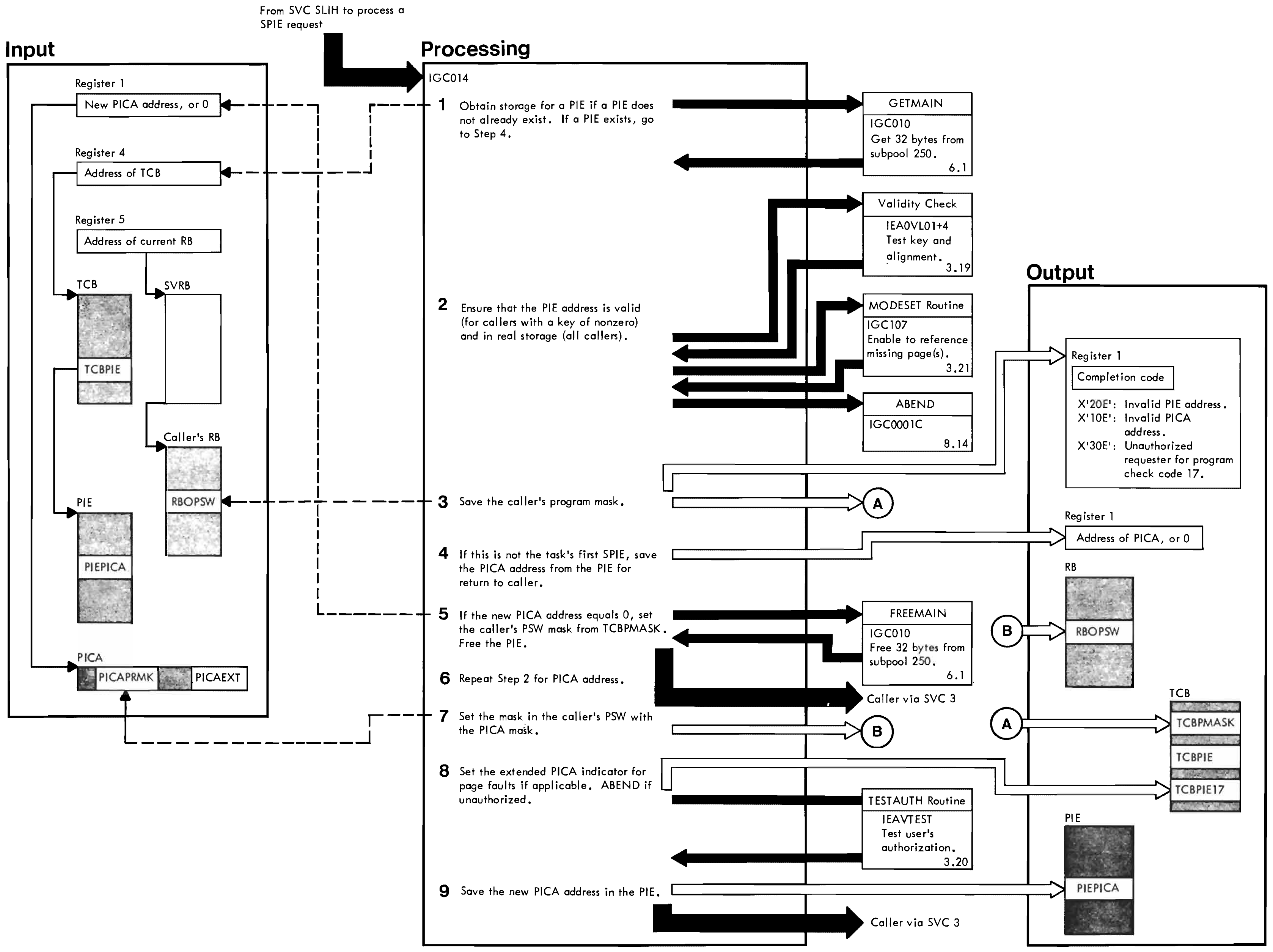
9 Schedule abnormal termination for tasks not already abnormally terminating (no STA exit allowed).

10 Wait for termination, then

X'33E': Subtask scheduled for abnormal termination and STA exit allowed to execute (STA retry not allowed).
X'13E': Subtask scheduled for abnormal termination. (STA exit not allowed).

Diagram 3.5 (Steps 7-10) DETACH Routine (Module IEAVED02)

NOTES	ROUTINE NAME	LABEL
8 If TCBIQE equals 0, then no ETXR exists. If the IRB use count is greater than 1, then the use count is decremented and the IRB is not freed.	DETACH	DTABN
9		STATEST
10		TESTSCND



1 Obtain storage for a PIE if a PIE does not already exist. If a PIE exists, go to Step 4.

2 Ensure that the PIE address is valid (for callers with a key of nonzero) and in real storage (all callers).

3 Save the caller's program mask.

4 If this is not the task's first SPIE, save the PICA address from the PIE for return to caller.

5 If the new PICA address equals 0, set the caller's PSW mask from TCBPMASK. Free the PIE.

6 Repeat Step 2 for PICA address.

7 Set the mask in the caller's PSW with the PICA mask.

8 Set the extended PICA indicator for page faults if applicable. ABEND if unauthorized.

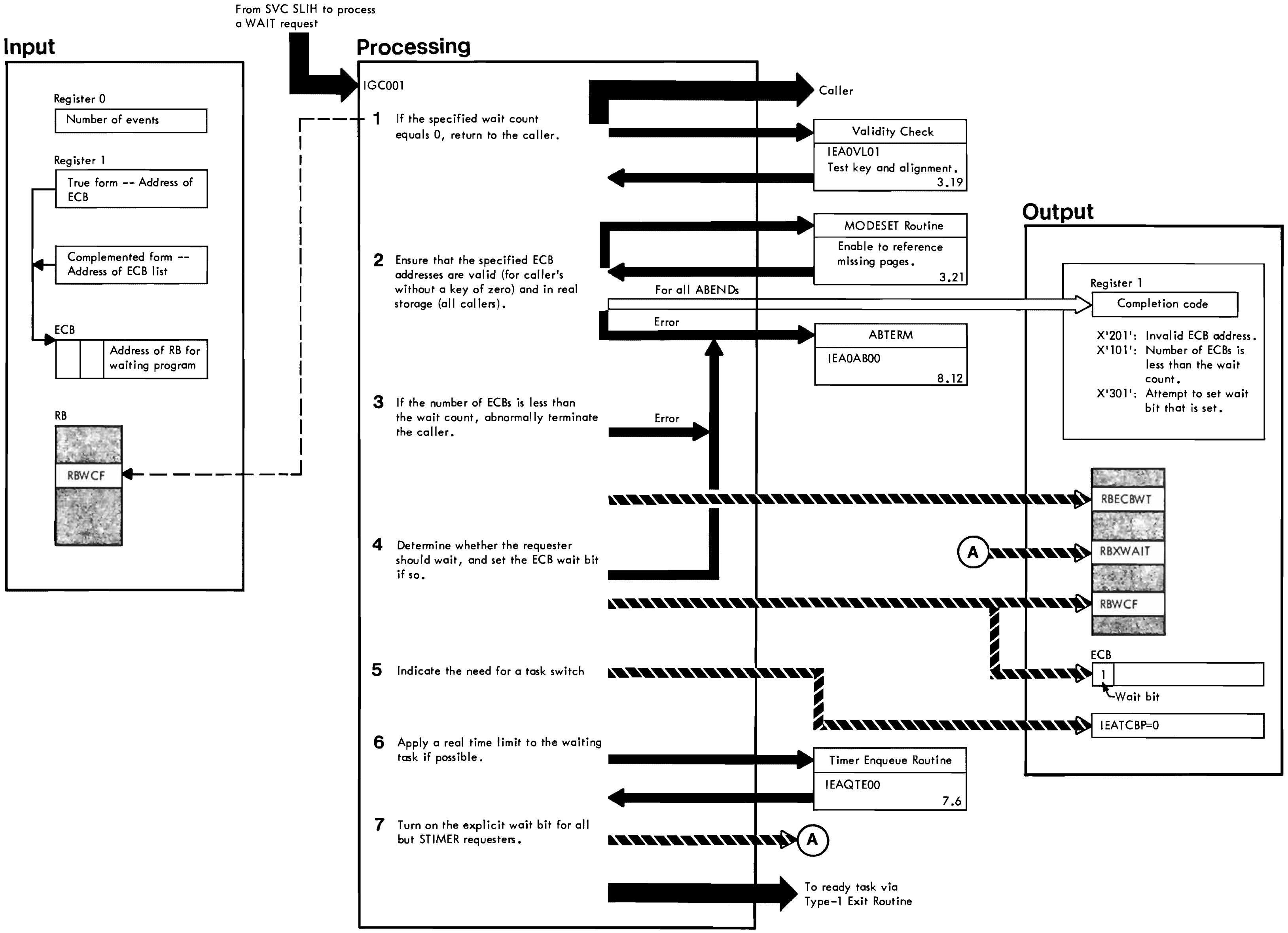
9 Save the new PICA address in the PIE.

• Diagram 3.6 SPIE Routine (Module IEAVTB00)

NOTES	ROUTINE NAME	LABEL
The SPIE routine 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 place a program mask, the address of the user's program-interruption exit routine, and an interruption mask in the fields of the PICA.	SPIE	IGC014
If, after the execution of the SPIE routine, a program-check interruption occurs in a program being executed for the issuer's task, the information in the PICA determines how the program interruption is to be processed.		
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 FLIH schedules abnormal termination of the task whose error caused the program interruption.		

NOTES	ROUTINE NAME	LABEL
1 If the TCBPIE field equals 0, this is the first time that the caller has issued a SPIE macro. A new PIE must be built.	SPIE	
2 For an already existing PIE, only the first word must be tested.		
4 If a PIE exists from the execution of an earlier SPIE, the PICA address it contains is returned to the caller in register 1. The caller may use this address in a later SPIE macro to restore this PICA.		PICASAV
5 Whenever a caller issues a SPIE macro with no operands, a zero PICA address results from the macro expansion. The saved program mask (TCBPMASK) is used to set the program mask in the caller's PSW. Thus, a SPIE macro with no operands cancels the effect of a previous SPIE macro.		RESPM
6 If the system is enabled at this point, processing must begin again at Step 2.		
If PICAEXT is not equal to zero, the TCBPIE17 bit is set equal to 1 if the user is authorized.		
8 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.		
9		SETNEWPC

Diagram 3.7
WAIT Routine


From SVC SLIH to process a WAIT request
Input
Register 0
Number of events
Register 1
True form -- Address of ECB
Complemented form -- Address of ECB list
ECB
Address of RB for waiting program
RB
RBWCF
Processing
IGC001
Caller
Validity Check
IEAOVL01
Test key and alignment.
3.19
MODESET Routine
Enable to reference missing pages.
3.21
For all ABENDs
Error
ABTERM
IEA0AB00
8.12
Error
A
Timer Enqueue Routine
IEAQTE00
7.6
A
To ready task via Type-1 Exit Routine
Output
Register 1
Completion code
X'201': Invalid ECB address.
X'101': Number of ECBs is less than the wait count.
X'301': Attempt to set wait bit that is set.
RBECBWT
RBXWAIT
RBWCF
ECB
1
Wait bit
IEATCBP=0

1 If the specified wait count equals 0, return to the caller.

2 Ensure that the specified ECB addresses are valid (for caller's without a key of zero) and in real storage (all callers).

3 If the number of ECBs is less than the wait count, abnormally terminate the caller.

4 Determine whether the requester should wait, and set the ECB wait bit if so.

5 Indicate the need for a task switch

6 Apply a real time limit to the waiting task if possible.

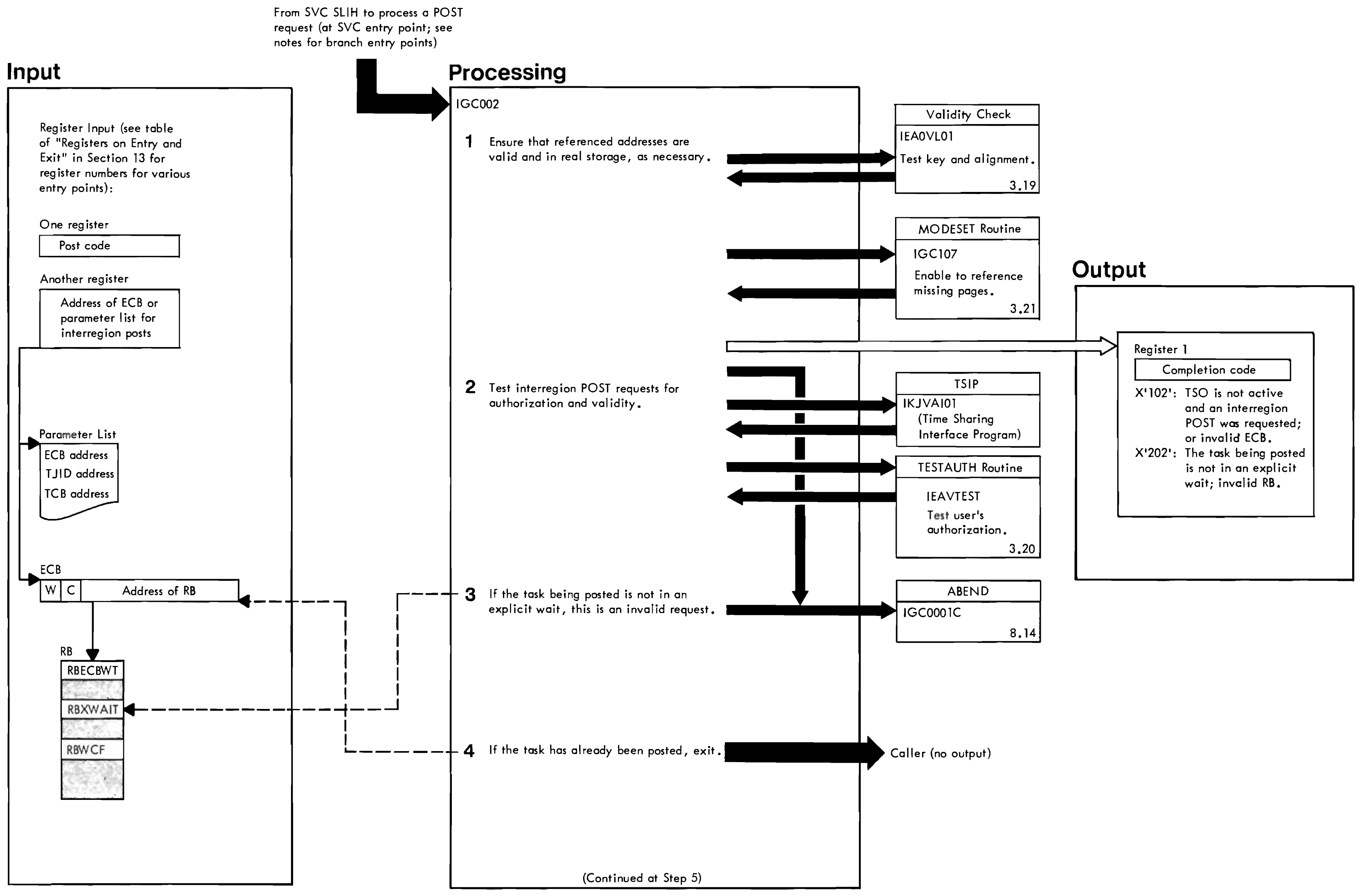
7 Turn on the explicit wait bit for all but STIMER requesters.

Diagram 3.7 WAIT Routine (Module IEAVSY50)

NOTES	ROUTINE NAME	LABEL
The WAIT routine 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 and makes the program ready (no longer waiting), so that its execution can continue.	WAIT	IGC001
3 The RBECBWT bit in the caller's RB is set when the caller specifies a smaller wait count than number of EBCs. 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. The misleading indicators are thus removed.		ECBWT
4		PKEY

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. • Return via SVC 3.
Yes	NA	≠0	• Continue processing ECBs. • Go to Step 5 if this is the last ECB.
No	Yes	NA	• ABTERM -- code X301.
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.

NOTES	ROUTINE NAME	LABEL
6 A time limit cannot be applied if: • A task in the subtask queue for the job step is not waiting. • The job-step task has no higher level task (TCBOTC = 0). • The task is a TSO task. • The caller has a key of 0. • There is no initiator TQE. • There is a real job-step TQE on the timer queue. • A task in the subtask queue for the job step has issued a STIMER macro in its highest-level RB. • WAIT has been called by the STIMER routine.	WAIT	JSTIME
7 When STIMER issues a WAIT macro instruction, it is requesting an interruption after a timed interval. It is not necessary for the WAIT routine to place additional time limits on the job step. A STIMER request is called an implicit wait. Other calls to the WAIT routine are explicit requests. The RBXWAIT bit is set to indicate explicit requests.		


From SVC SLIH to process a POST request (at SVC entry point; see notes for branch entry points)
Input
Register Input (see table of "Registers on Entry and Exit" in Section 13 for register numbers for various entry points):
One register
Post code
Another register
Address of ECB or parameter list for interregion posts
Parameter List
ECB address
TJID address
TCB address
ECB
W
C
Address of RB
RB
RBECBWT
RBXWAIT
RBWCF
Processing
IGC002
Validity Check
IEAOVL01
Test key and alignment.
3.19
MODESET Routine
IGC107
Enable to reference missing pages.
3.21
TSIP
IKJVAI01
(Time Sharing Interface Program)
TESTAUTH Routine
IEAVTEST
Test user's authorization.
3.20
ABEND
IGC0001C
8.14
Caller (no output)
Output
Register 1
Completion code
(Continued at Step 5)

1 Ensure that referenced addresses are valid and in real storage, as necessary.

2 Test interregion POST requests for authorization and validity.

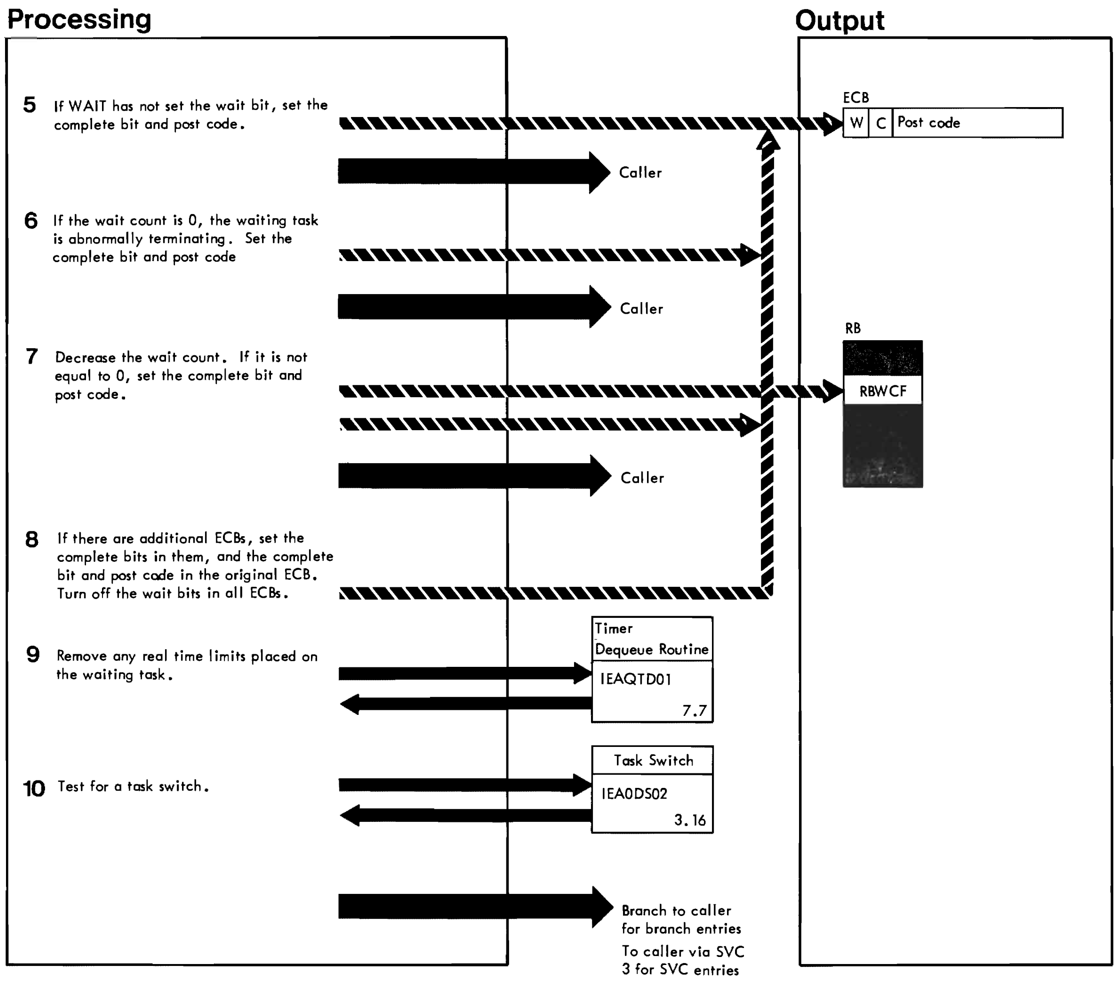
3 If the task being posted is not in an explicit wait, this is an invalid request.

4 If the task has already been posted, exit.

X'102':	TSO is not active and an interregion POST was requested; or invalid ECB.
X'202':	The task being posted is not in an explicit wait; invalid RB.

Diagram 3.8 (Steps 1-4) POST Routine (Module IEAVSY50)

NOTES	ROUTINE NAME	LABEL
The POST routine permits a program (the "posting" program or caller) to signal the occurrence of an event, such as the completion of an I/O operation, awaited by a waiting program. The routine signals (posts) the event's occurrence by altering a bit in a specified ECB (event control block) shared by both the waiting and posting programs.	POST	IGC002
The POST routine also places in the event control block a post code supplied by the posting program. The post code may later be inspected by the waiting program, after it resumes execution, to determine the type of event that occurred. The POST routine determines whether the program that is awaiting the posted event can be made ready (whether all awaited events have occurred).		
If the waiting program can be made ready, and belongs to a task of higher dispatching priority than that of the posting program, the POST routine indicates to the dispatcher that a task switch is needed (a ready program having higher priority than that of the caller should be dispatched). If a time limit has been applied to the waiting task, it is removed.		
1 After branch entries and interregion SVC entries (see table below), POST returns control to the caller without completing its processing if it is necessary to enable to bring the pages into storage. Before returning to the caller, POST provides for an asynchronous reentry to itself. It does this by calling GETMAIN to get storage for an SQE (supervisor queue element), initializing the SQE, and calling the Stage 2 Exit Effector to begin scheduling the asynchronous reentry. This reentry will accomplished under the task being posted. The Stage 2 Exit Effector queues the SQE on an asynchronous exit queue (see Diagram 3.9).		
Asynchronous reentry protects supervisor routines from time-consuming processing involved in ensuring that pages are in real storage. In addition, many of POST's callers could not tolerate an enabling of the system. With asynchronous reentry, the waiting routine, not the caller of POST, will incur the missing page interruption.		
2 If a TSO task is being posted, the TJB is referenced to determine whether the user is in storage. For the active user, processing continues. If the task is swapped out, an IPPB (interpartition POST block) is created, and the TSIP (Time sharing Interface Program) is called to bring the task into storage. POST is then called by the RCT (Region Control Task).		TSTJID
4		POSTTEST



Processing

5 If WAIT has not set the wait bit, set the complete bit and post code.

6 If the wait count is 0, the waiting task is abnormally terminating. Set the complete bit and post code

7 Decrease the wait count. If it is not equal to 0, set the complete bit and post code.

8 If there are additional ECBs, set the complete bits in them, and the complete bit and post code in the original ECB. Turn off the wait bits in all ECBs.

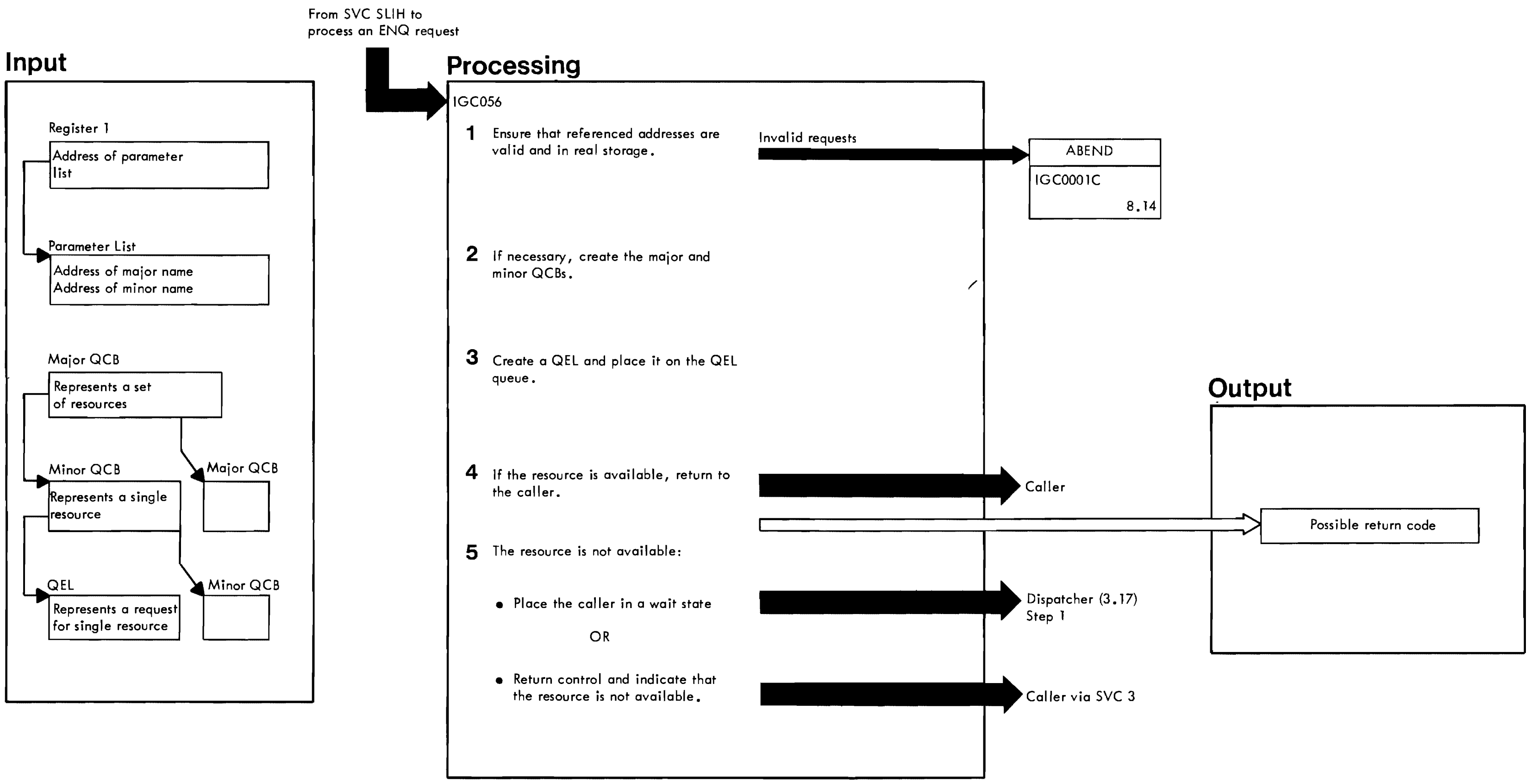
9 Remove any real time limits placed on the waiting task.

10 Test for a task switch.

Output

Diagram 3.8 (Steps 5-10) POST Routine (Module IEAVSY50)

NOTES	ROUTINE NAME	LABEL
8 If processing of the ECB list causes the system to be enabled, POST must begin again at Step 1.	POST	
10 On branch entries, POST does not effect a task switch, even if one is indicated. Control returns to the caller, and no task switch occurs until the dispatcher is entered again.		



1 Ensure that referenced addresses are valid and in real storage.

2 If necessary, create the major and minor QCBs.

3 Create a QEL and place it on the QEL queue.

4 If the resource is available, return to the caller.

5 The resource is not available:

- Place the caller in a wait state

OR

- Return control and indicate that the resource is not available.

Diagram 3.9 ENQ Routine (Module IEAVENQ1)

NOTES	ROUTINE NAME	LABEL
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 storage. The symbolic name of the resource, not the actual resource itself, is used by ENQ to control access. 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 routine and the dispatcher, returns control to the caller, which then gains access to its resource(s). But if any requested resource is already in use by another ENQ-issuing program, being executed for another task, the ENQ routine may place the caller in a wait condition until the resource becomes available.	ENQ	IGC056

2 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.		ENDMAJ
		ENDMIN

3		CREATQEL
4 If the requested resource is not in use, as indicated by the absence of QCBs with the specified Qname and Rname, control is returned to the caller. Depending on the input to the service routine, a return code may or may not be issued, and a QEL may or may not be constructed and placed on the resource queues. (Refer to Figure 3-5 for the various results.)		TESTEND

NOTES	ROUTINE NAME	LABEL
5 If another requester has access to the resource, as indicated by a major and minor QCB containing the resource names, resultant 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. (Figure 3-6 lists the different forms of resultant processing.) Note in Figure 3-6 that a QEL is constructed and placed on a QEL queue if the requester wants access to the resource and is willing to wait for it. 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 (see Figure 3-7). If RET is USE, a QEL is created only if the requester can have immediate access to the resource. Note that if all previous QELs on the queue and the present request are both for "shared" resources, there is no need for the caller to wait. The new requester and those represented by the "shared" previous QELs on the queue may share the resource on a task-priority basis. Thus, a requester need not have its QEL at the top of the "shared" group of QELs. Any requester represented in the shared group may be executed if other requesters represented in the group are waiting for an event, such as an I/O completion, provided at least one member of the group is at the top of the queue. Control is returned to the caller if the requested resource or resources are available, or to the current routine of the next highest priority ready task if the caller must wait because the requested resource is in use. If the caller is to receive control, the return path is via the Exit routine and the dispatcher. But if the current routine of another task is to receive control, the return path is via the dispatcher only. To determine the appropriate return path, ENQ tests the RB wait count field in the current SVRB. If the RB wait count is 0, all requested resources are available and the caller can receive control. But if the RB wait count is greater than 0, the caller is effectively in a wait condition and cannot be given control.	ENQ	ERRX5 RET12

RET Parameter	Meaning of RET Parameter	QCB and/or QEL Constructed and Queued	Control is Returned to Caller With Code of:	Meaning of Return Code
TEST	Tests the queues to determine if the caller can have immediate use of the resource. Never constructs control blocks.	no	0	Resource is available
USE	Places QCB and/or QEL on queues only if caller can have immediate access to the resource.	yes	0	Resource is available
HAVE	Delay can be tolerated. Places QCB and/or QEL on queues.	yes	0	Resource is available
NONE or omitted	Same as HAVE but produces no return code.	yes	no code	
CHNG	Converts a shared QEL to exclusive if (a) the QEL is shared (b) the QEL has control of the resource & (c) no other QEL is sharing.	no	8	No QEL existed to be converted

Figure 3-5. Processing if a requested resource is not in use.

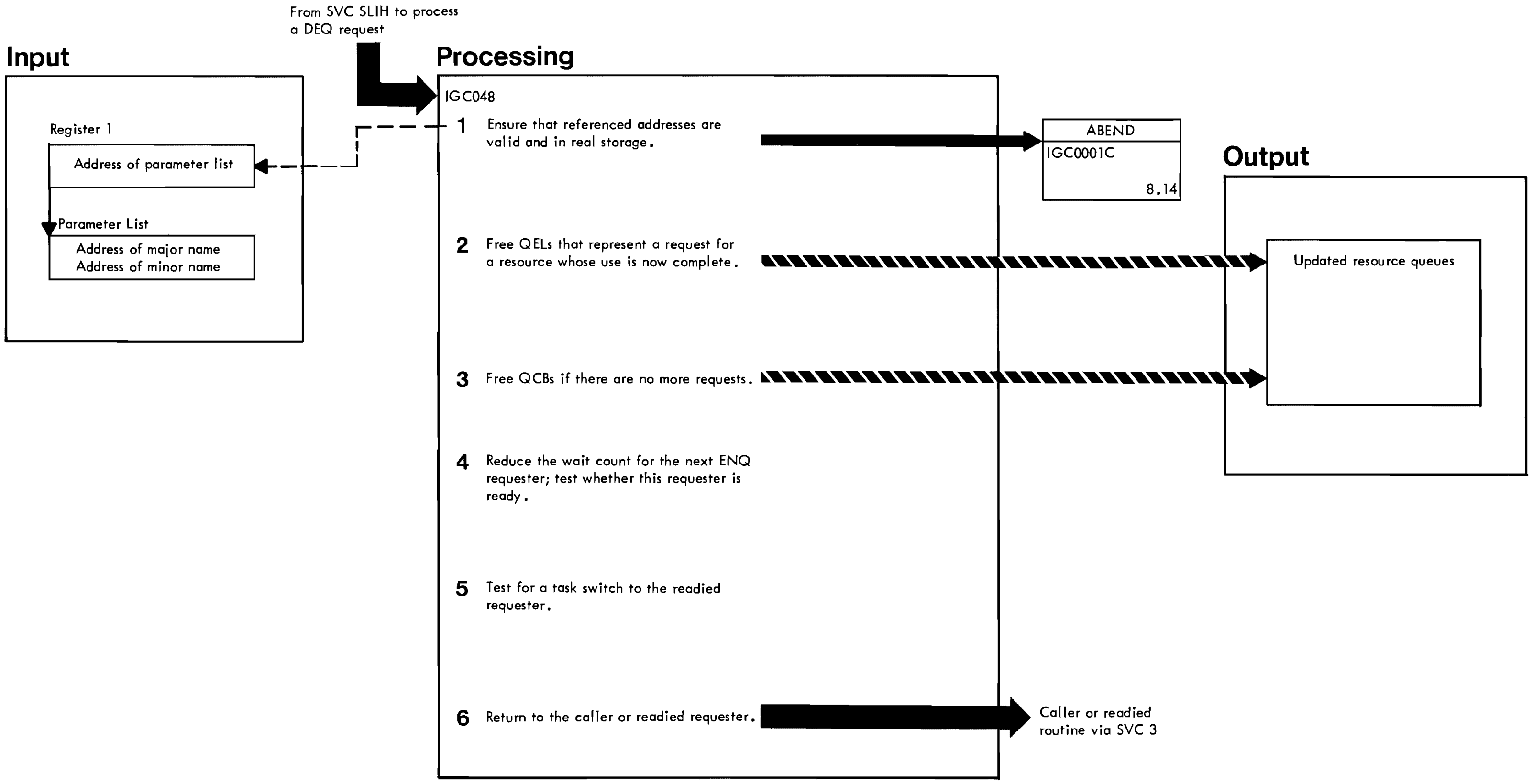
Type of Previous QEL and Present Request	RET parameter is:	Resultant Processing
1. The previous QEL on the queue is "exclusive," or the present request is "exclusive"	USE or TEST	Sets return code equal to 4 and, via the Exit routine and the Dispatcher, returns control to the caller. A QEL is not constructed to represent the request.
	HAVE, NONE or omitted	Places requester into wait condition by increasing SVRB wait count, constructs a QEL and places it on a QEL queue, indicates that a task switch is needed, branches to the Dispatcher to perform a task switch. If RET is HAVE, a return code of 0 is also produced and passed to requester after resource becomes available.
2. The previous QELs and the present request are both "shared," <u>or</u>	TEST	Sets return code equal to 0 and, via the Exit routine and the Dispatcher, returns control to the caller. No QEL is constructed.
There is no previous QEL for the resource (that is, the QEL queue is empty)	NONE or omitted	Constructs a new QEL, places it on a QEL queue, and via the Exit routine and the Dispatcher, returns control to the caller.
	USE or HAVE	Sets return code equal to 0, constructs a new QEL, places it on a QEL queue, and via the Exit routine and the Dispatcher, returns control to the caller.
3. The caller does not control and resource or is actively sharing	CHNG	Sets return code equal to 4 and, via the Exit routine and the Dispatcher, returns control to the caller. No QEL is constructed.
4. The caller has control of resource and QEL is shared with no other QEL sharing	CHNG	Set return code equal to 0, changes QEL to Exclusive and, via the Exit routine and Dispatcher, returns control to the caller.

Figure 3-6. Processing if a requested resource is in use.

Return Code	RET=TEST,USE,HAVE	RET=CHNG	ECB=
20	The caller's task is already en-queued but does not have control of resource.	Not used	The caller's task is already enqueued but does not have control of resource. No QEL created.
16	Not used	Not used	Previous ENQ with ECB request still pending.
12	Resource is permanently unavailable.	Resource is permanently unavailable.	Resource is permanently unavailable. No QEL created.
8	The caller's task is already en-queued and has control of resource.	No QEL exists for the caller.	Caller's task is already enqueued and has control of resource. No QEL created.
4	The resource is in use, and the caller's request has not been en-queued. (Only TEST & USE.)	The caller does not have sole control of resource.	Resource is in use and caller's request has been placed on queue.
0	The resource is available, or the caller's request has been enqueued.	Resource is now under exclusive control.	Resource is available and caller's request has been enqueued.

Figure 3-7. Return codes for the ENQ routine.

Diagram 3.10 DEQ Routine



1 Ensure that referenced addresses are valid and in real storage.

2 Free QELs that represent a request for a resource whose use is now complete.

3 Free QCBs if there are no more requests.

4 Reduce the wait count for the next ENQ requester; test whether this requester is ready.

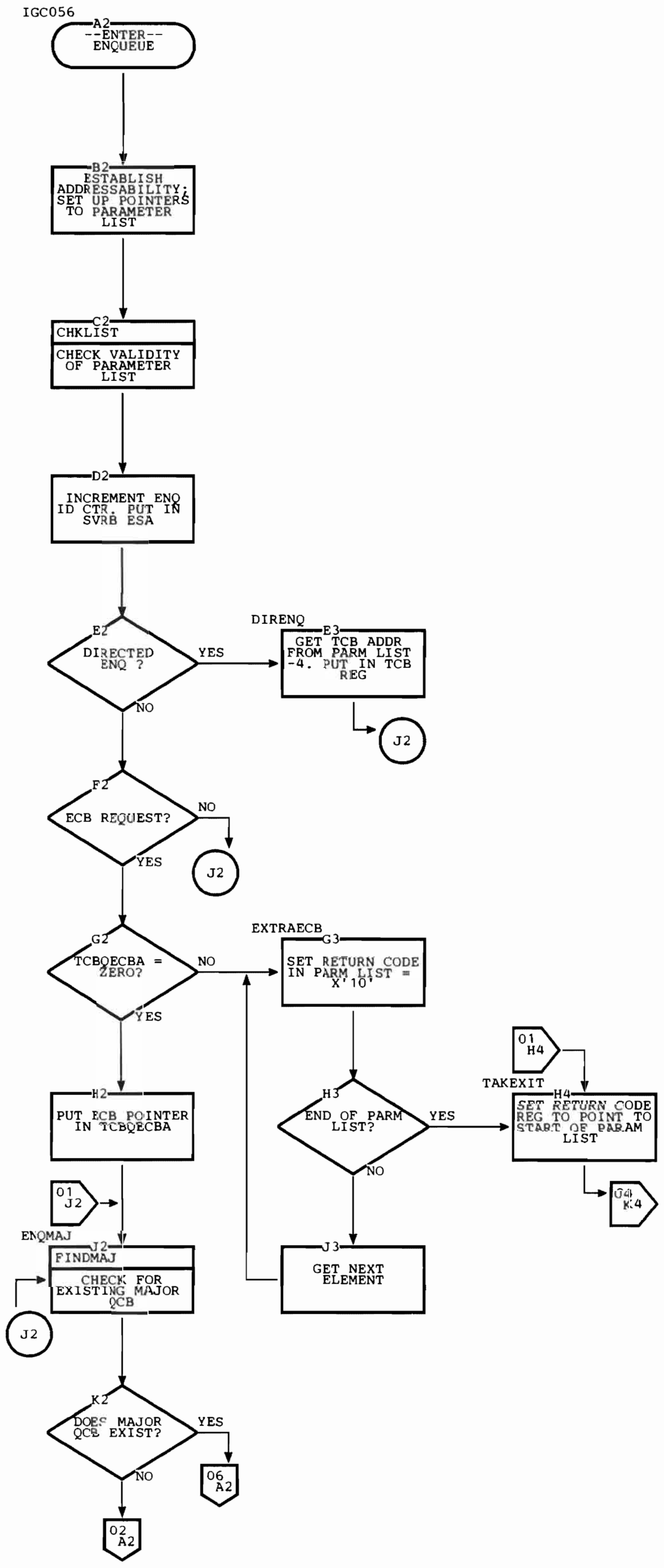
5 Test for a task switch to the readied requester.

6 Return to the caller or readied requester.

Diagram 3.10 DEQ Routine (Module IEAVENQ1)

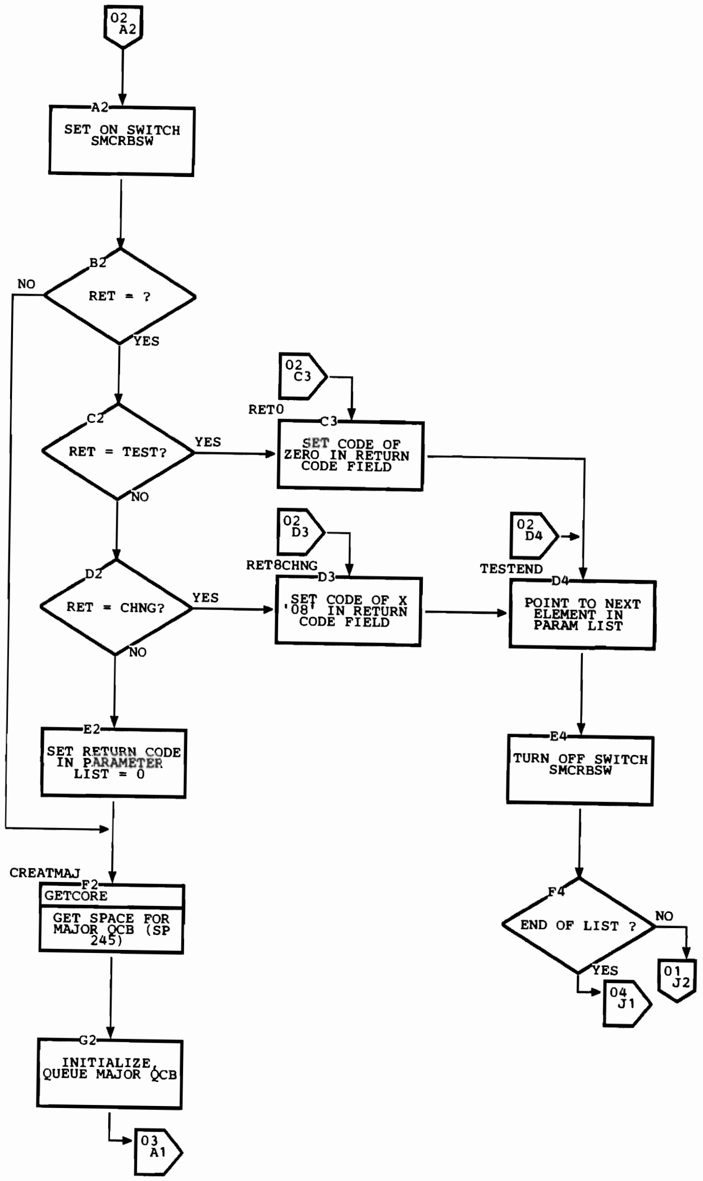
NOTES	ROUTINE NAME	LABEL
When the program finishes using the resource(s), it issues a DEQ macro instruction which causes the DEQ routine to remove one or more elements from the request queue and to reduce the wait count for the waiting program. If the wait count is now 0, the DEQ routine, via the Exit routine and the dispatcher, may return control to the previously waiting (now ready) program. The program then gains access to its resource(s).	DEQ	IGC048
2 To update the resource queues, 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. 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 logically at the top of its queue, DEQ dequeues the QEL and, via supervisor linkage to FREEMAIN, frees the space that the QEL occupies.		PARMLOOP
3 DEQ examines the QCB queues to determine if any QCB may be released. If there are no more QELs queued to the minor QCB for the resource, there are no further requests for the resource, and the minor QCB can be released. In this case, DEQ removes the minor QCB from its queue and frees the space it occupies. It then examines the minor QCB queue to decide whether the minor QCB is needed and can be similarly eliminated. If there are no minor QCBs queued to the major QCB, there are no outstanding requests for the entire set of resources. In this case, DEQ removes the major QCB from its queue and frees its space. DEQ then processes in a similar manner any other input parameters that represent QELs to be dequeued.		PROCMIN
4 For each SVRB whose awaited resources are available, as indicated by a zero RB wait count, the DEQ routine tests whether the associated requester can be dispatched. If the requester's task is of higher dispatching priority than the caller's, the requester may be dispatched in place of the caller.		LOOP1 REDUCE
5 For each SVRB that has a zero wait count, DEQ calls the supervisor's Task Switch routine to compare dispatching priorities. If the readied requester's TCB has a higher priority than the caller's, the Task Switch routine indicates this fact to the dispatcher by placing the requester's TCB address in the "new" TCB pointer, IEATCBP.		TASKWTH
6 DEQ returns control to the caller or a readied requester, via the Exit routine and the dispatcher. The Exit routine dequeues the SVRB from its RB queue and frees the space that the SVRB occupies. The dispatcher decides whether to return control to the caller or to a readied requester, depending on the contents of the "new" TCB pointer, IEATCBP. If the "new" TCB pointer contains the address of the current TCB, the dispatcher returns control to the caller. Otherwise, the dispatcher returns control to the requester whose TCB address is in the pointer. In this case a task switch has occurred.		

Chart 3-1 (page 1 of 25). **ENQ/DEQ/RESERVE**



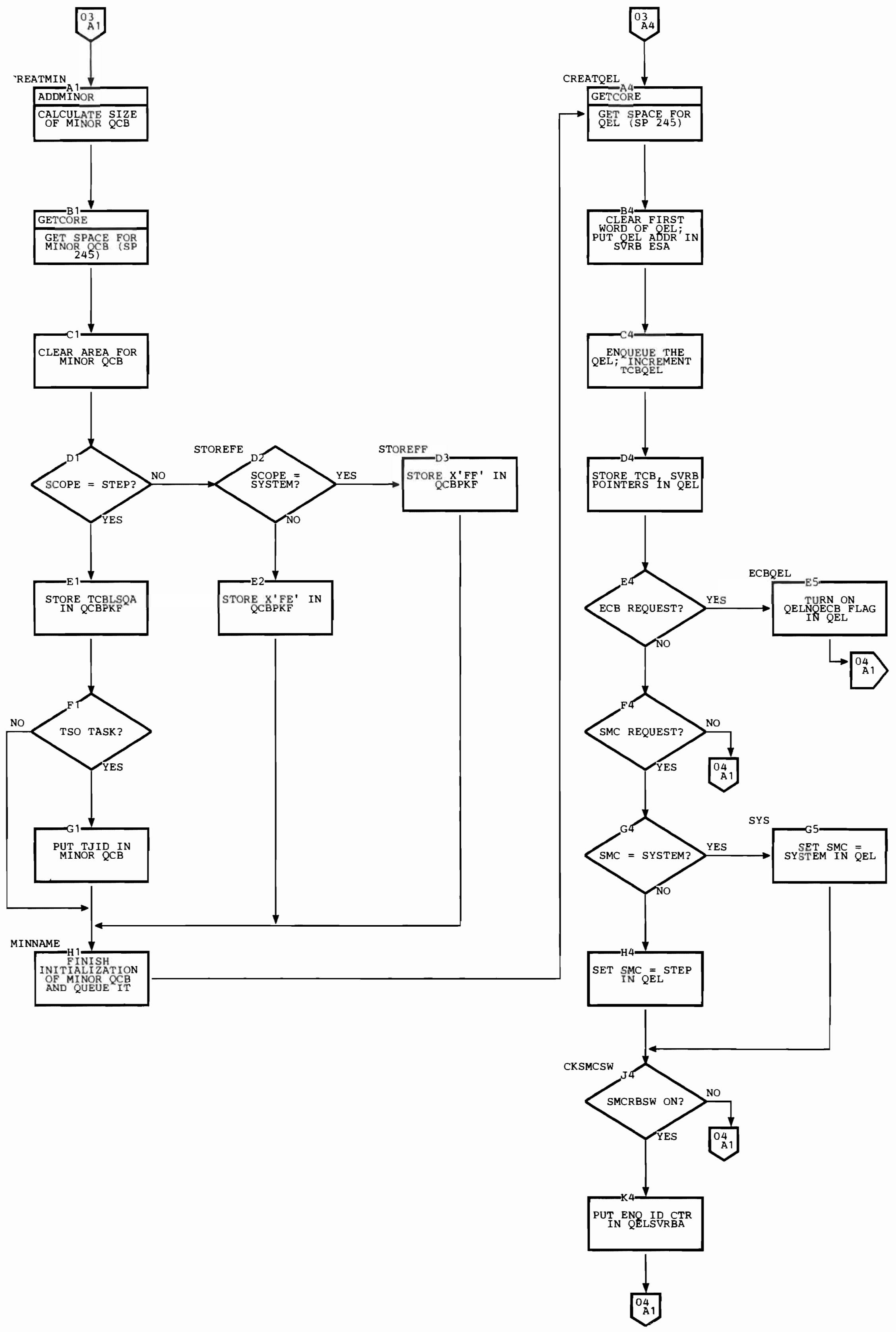
IGC056
A2 --ENTER-- ENQUEUE
B2 ESTABLISH ADDRESSABILITY; SET UP POINTERS TO PARAMETER LIST
C2 CHKLIST
CHECK VALIDITY OF PARAMETER LIST
D2 INCREMENT ENQ ID CTR. PUT IN SVRB ESA
E2 DIRECTED ENQ ?
YES
NO
DIRENQ
E3 GET TCB ADDR FROM PARM LIST -4. PUT IN TCB REG
J2
F2 ECB REQUEST?
NO
YES
J2
G2 TCBQECBA = ZERO?
NO
YES
EXTRAECB
G3 SET RETURN CODE IN PARM LIST = X'10'
H2 PUT ECB POINTER IN TCBQECBA
H3 END OF PARM LIST?
YES
NO
01 H4
TAKEXIT
H4 SET RETURN CODE REG TO POINT TO START OF PARAM LIST
04 K4
01 J2
ENQMAJ
J2 FINDMAJ
CHECK FOR EXISTING MAJOR QCB
J2
J3 GET NEXT ELEMENT
K2 DOES MAJOR QCB EXIST?
YES
NO
06 A2
02 A2

Chart 3-1 (page 2 of 25). **ENQ/DEQ/RESERVE**



02 A2
A2
SET ON SWITCH SMCRBSW
B2
RET = ?
NO
YES
02 C3
RET0
C3
C2
RET = TEST?
YES
SET CODE OF ZERO IN RETURN CODE FIELD
NO
02 D3
RET8CHNG
D3
02 D4
TESTEND
D4
D2
RET = CHNG?
YES
SET CODE OF X '08' IN RETURN CODE FIELD
POINT TO NEXT ELEMENT IN PARAM LIST
NO
E2
SET RETURN CODE IN PARAMETER LIST = 0
E4
TURN OFF SWITCH SMCRBSW
CREATMAJ
F2
GETCORE
GET SPACE FOR MAJOR QCB (SP 245)
F4
END OF LIST ?
NO
YES
01 J2
04 J1
G2
INITIALIZE, QUEUE MAJOR QCB
03 A1

Chart 3-1 (page 3 of 25). ENQ/DEQ/RESERVE



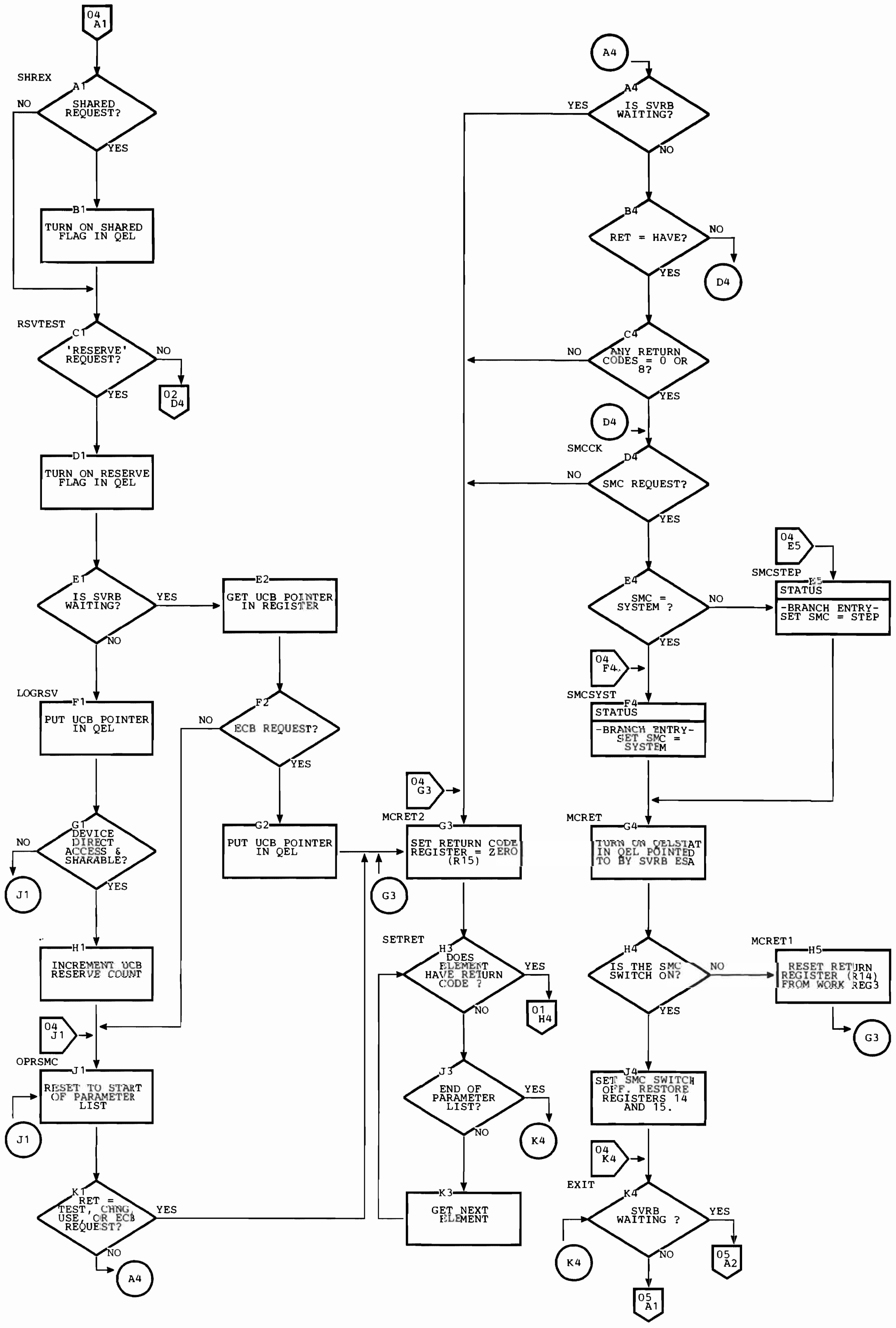
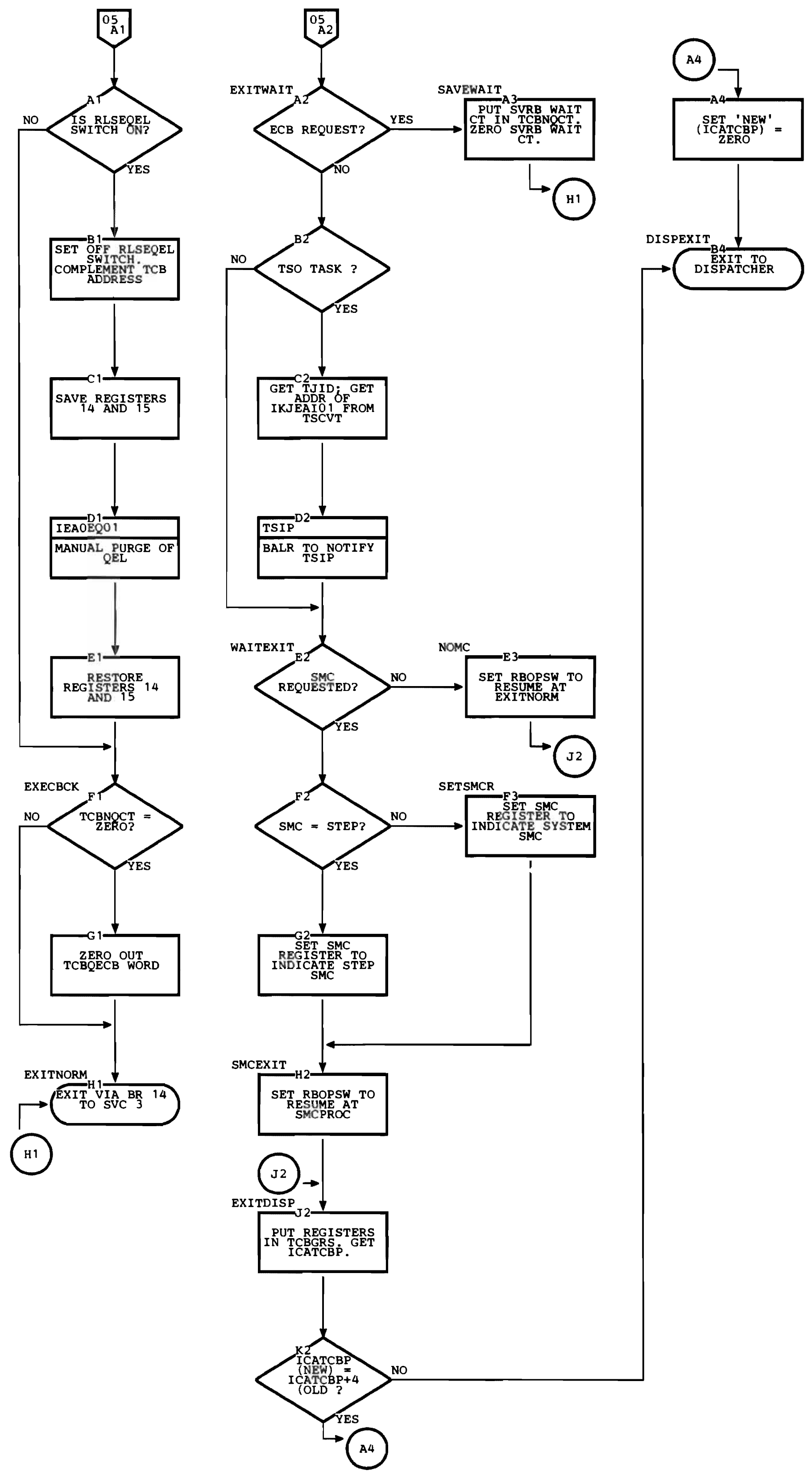
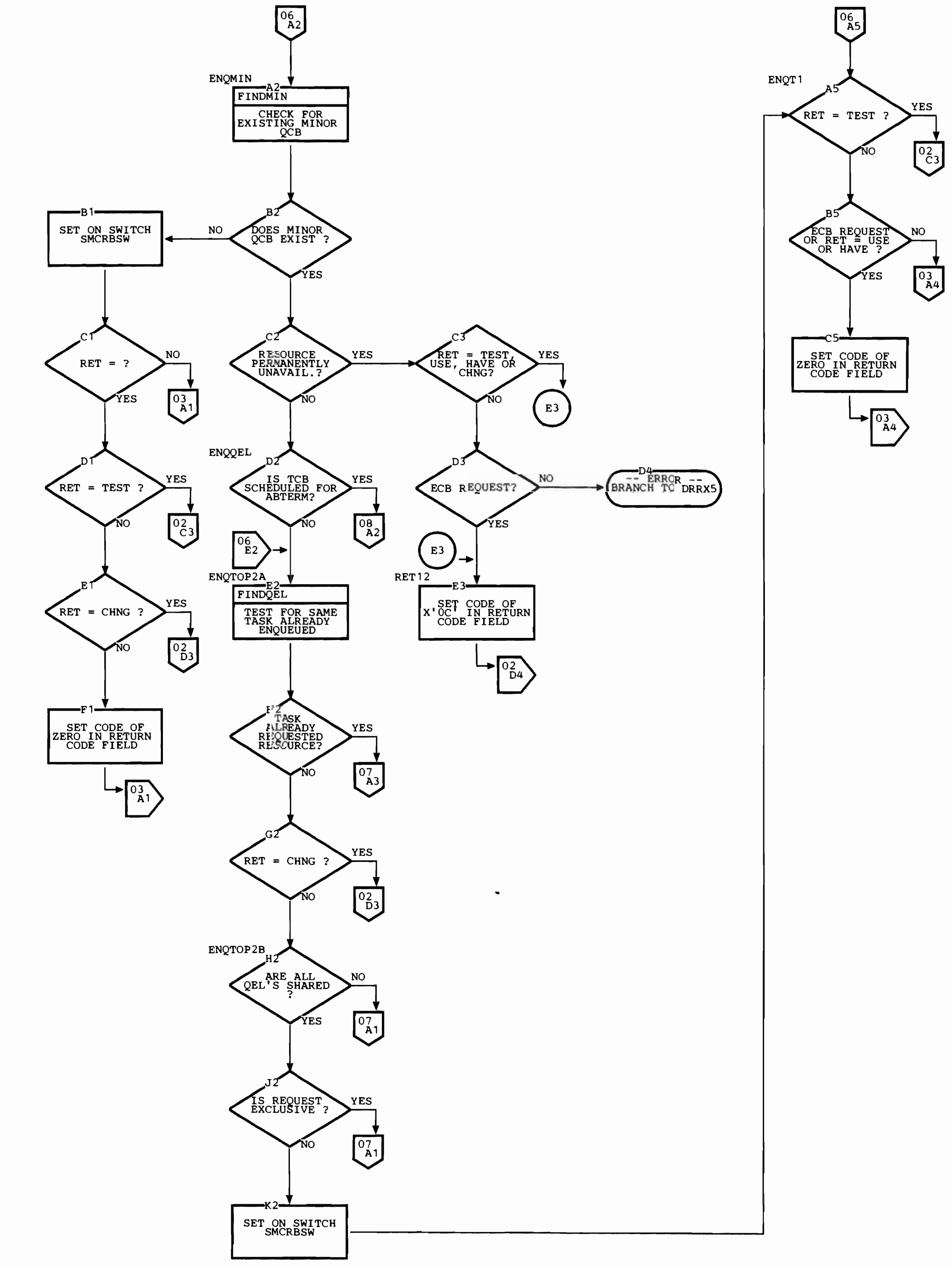

04 A1
SHREX
A1 SHARED REQUEST?
NO
YES
B1 TURN ON SHARED FLAG IN QEL
RSVTEST
C1 'RESERVE' REQUEST?
02 D4
D1 TURN ON RESERVE FLAG IN QEL
E1 IS SVRB WAITING?
E2 GET UCB POINTER IN REGISTER
LOGRSV
F1 PUT UCB POINTER IN QEL
F2 ECB REQUEST?
G1 DEVICE DIRECT ACCESS & SHARABLE?
G2 PUT UCB POINTER IN QEL
J1
H1 INCREMENT UCB RESERVE COUNT
04 J1
OPRSMC
J1 RESET TO START OF PARAMETER LIST
K1 RET = TEST, CHNG, USE, OR ECB REQUEST?
A4
04 G3
MCRET2
G3 SET RETURN CODE REGISTER = ZERO (R15)
G3
SETRET
H3 DOES ELEMENT HAVE RETURN CODE ?
01 H4
J3 END OF PARAMETER LIST?
K4
K3 GET NEXT ELEMENT
A4
A4 IS SVRB WAITING?
B4 RET = HAVE?
D4
C4 ANY RETURN CODES = 0 OR 8?
SMCCK
D4 SMC REQUEST?
04 E5
SMCSTEP
E5 STATUS -BRANCH ENTRY- SET SMC = STEP
E4 SMC = SYSTEM ?
04 F4
SMCSYST
F4 STATUS -BRANCH ENTRY- SET SMC = SYSTEM
MCRET
G4 TURN ON QELSTAT IN QEL POINTED TO BY SVRB ESA
H4 IS THE SMC SWITCH ON?
MCRET1
H5 RESET RETURN REGISTER (R14) FROM WORK REG3
J4 SET SMC SWITCH OFF. RESTORE REGISTERS 14 AND 15.
04 K4
EXIT
K4 SVRB WAITING ?
05 A2
05 A1

Chart 3-1 (page 5 of 25). **ENQ/DEQ/RESERVE**



05 A1
05 A2
A1 IS RLSEQEL SWITCH ON?
NO
YES
B1 SET OFF RLSEQEL SWITCH. COMPLEMENT TCB ADDRESS
C1 SAVE REGISTERS 14 AND 15
D1 IEAOEQ01 MANUAL PURGE OF QEL
E1 RESTORE REGISTERS 14 AND 15
EXECBCK F1 TCBNQCT = ZERO?
G1 ZERO OUT TCBQECB WORD
EXITNORM H1 EXIT VIA BR 14 TO SVC 3
H1
EXITWAIT A2 ECB REQUEST?
SAVEWAIT A3 PUT SVRB WAIT CT IN TCBNQCT. ZERO SVRB WAIT CT.
B2 TSO TASK ?
C2 GET TJID; GET ADDR OF IKJEAI01 FROM TSCVT
D2 TSIP BALR TO NOTIFY TSIP
WAITEXIT E2 SMC REQUESTED?
NOMC E3 SET RBOPSW TO RESUME AT EXITNORM
J2
F2 SMC = STEP?
SETSMCR F3 SET SMC REGISTER TO INDICATE SYSTEM SMC
G2 SET SMC REGISTER TO INDICATE STEP SMC
SMCEXIT H2 SET RBOPSW TO RESUME AT SMCPROC
EXITDISP J2 PUT REGISTERS IN TCBGRS. GET ICATCBP.
K2 ICATCBP (NEW) = ICATCBP+4 (OLD ?
A4
A4 SET 'NEW' (ICATCBP) = ZERO
DISPEXIT B4 EXIT TO DISPATCHER

Chart 3-1 (page 6 of 25). ENQ/DEQ/RESERVE



06 A2
ENQMIN
A2 FINDMIN
CHECK FOR EXISTING MINOR QCB
B2 DOES MINOR QCB EXIST ?
NO
YES
B1 SET ON SWITCH SMCRBSW
C1 RET = ?
NO
YES
03 A1
D1 RET = TEST ?
YES
NO
02 C3
E1 RET = CHNG ?
YES
NO
02 D3
F1 SET CODE OF ZERO IN RETURN CODE FIELD
03 A1
C2 RESOURCE PERMANENTLY UNAVAIL.?
YES
NO
C3 RET = TEST, USE, HAVE OR CHNG?
YES
NO
E3
ENQQEL
D2 IS TCB SCHEDULED FOR ABTERM?
YES
NO
08 A2
D3 ECB REQUEST?
NO
YES
D4 -- ERROR -- BRANCH TO DRRX5
06 E2
ENQTOP2A
E2 FINDQEL
TEST FOR SAME TASK ALREADY ENQUEUED
E3
RET12
E3 SET CODE OF X'0C' IN RETURN CODE FIELD
02 D4
F2 TASK ALREADY REQUESTED RESOURCE?
YES
NO
07 A3
G2 RET = CHNG ?
YES
NO
02 D3
ENQTOP2B
H2 ARE ALL QEL'S SHARED ?
NO
YES
07 A1
J2 IS REQUEST EXCLUSIVE ?
YES
NO
07 A1
K2 SET ON SWITCH SMCRBSW
06 A5
ENQT1
A5 RET = TEST ?
YES
NO
02 C3
B5 ECB REQUEST OR RET = USE OR HAVE ?
NO
YES
03 A4
C5 SET CODE OF ZERO IN RETURN CODE FIELD
03 A4

Chart 3-1 (page 7 of 25). ENQ/DEQ/RESERVE

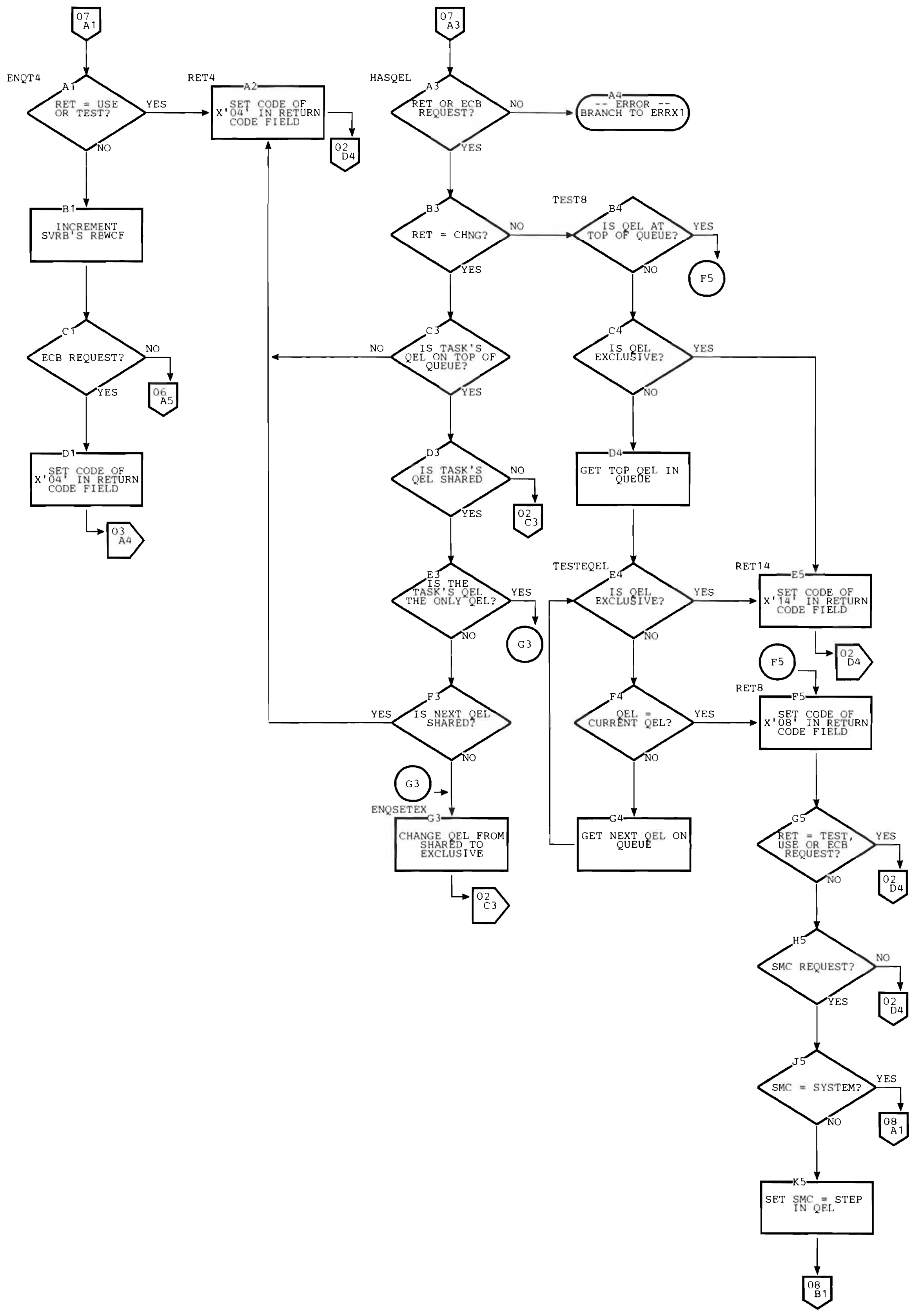


Chart 3-1 (page 8 of 25). ENQ/DEQ/RESERVE

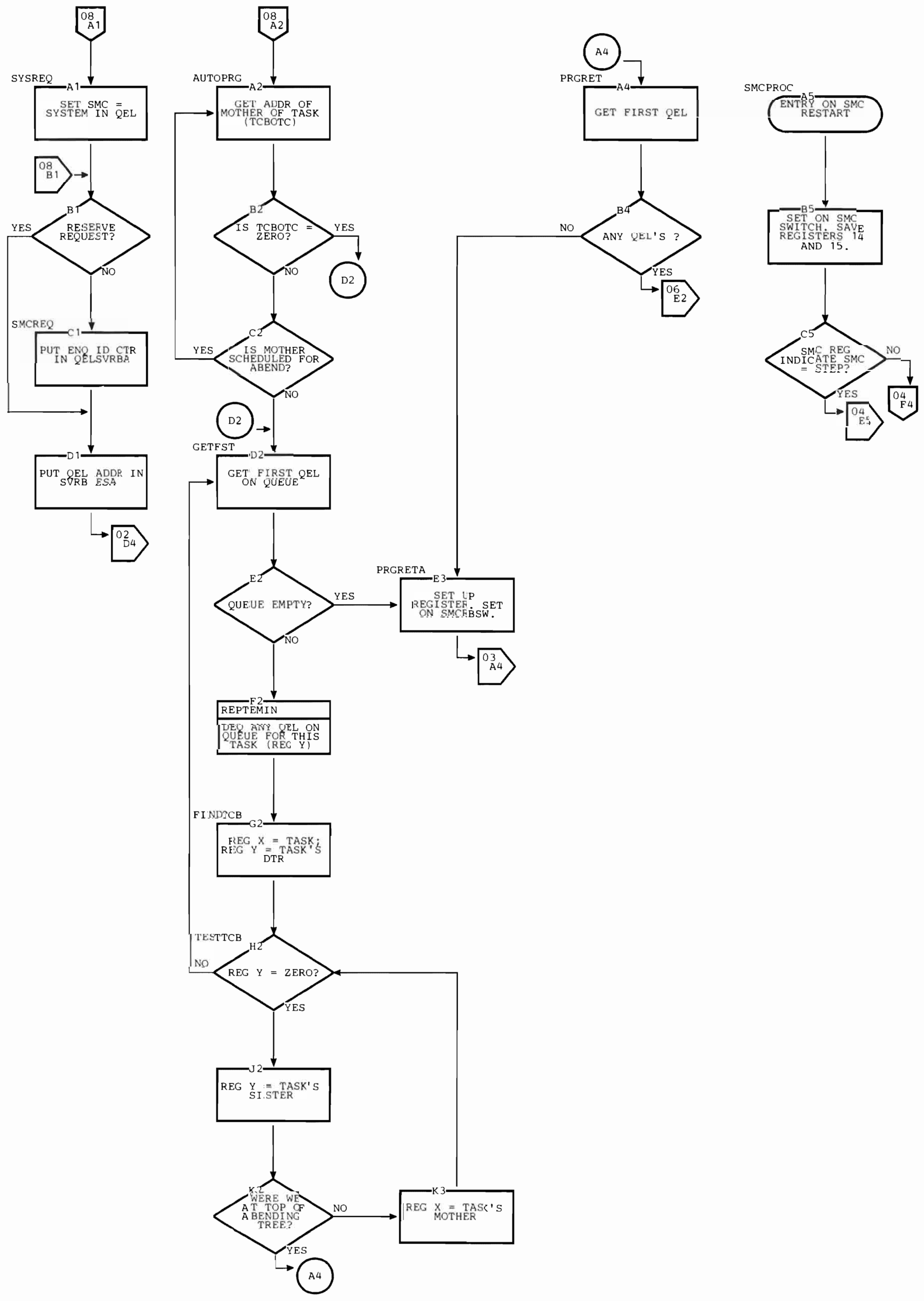
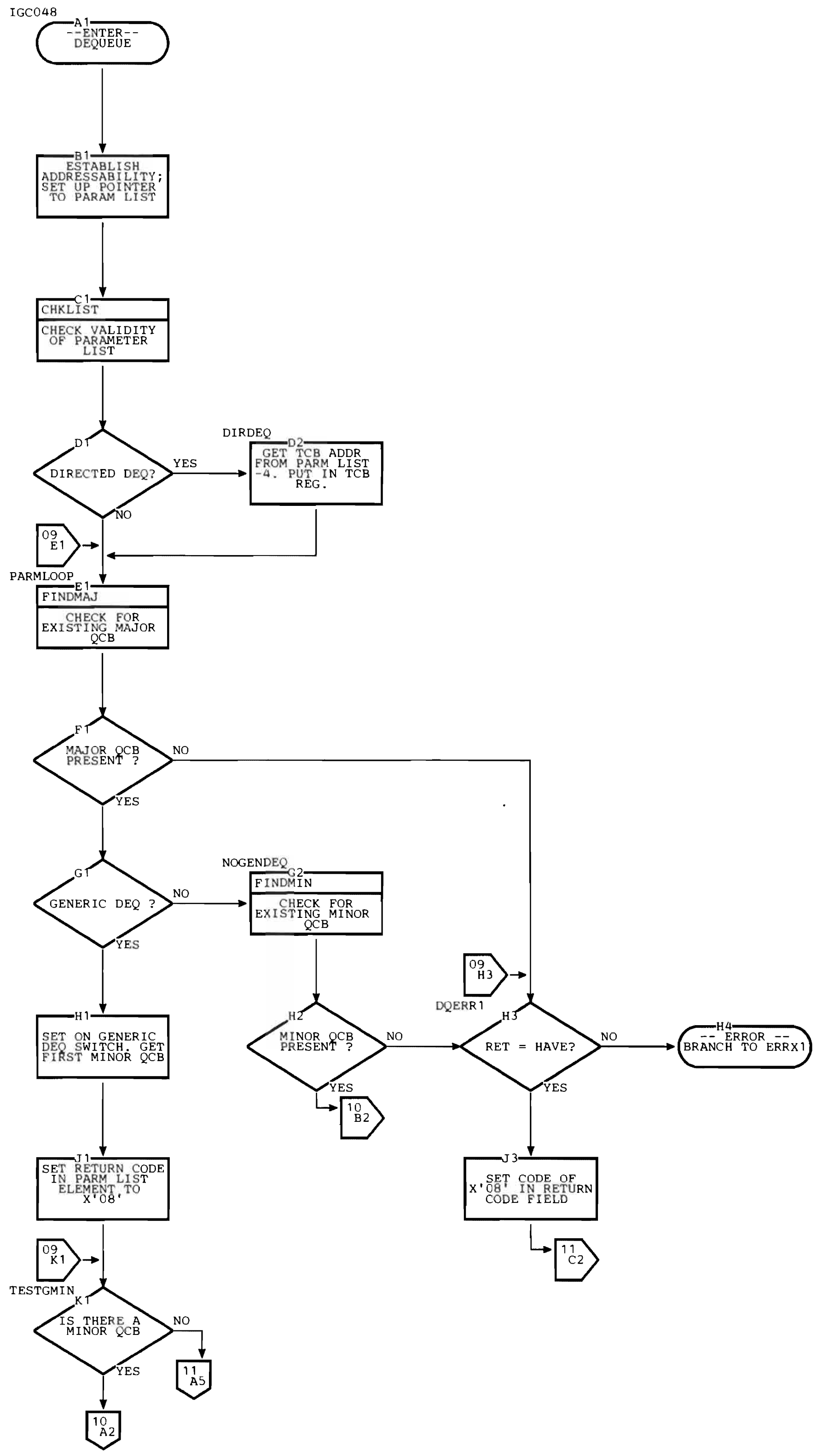

08 A1
SYSREQ A1 SET SMC = SYSTEM IN QEL
08 B1
B1 RESERVE REQUEST?
YES
NO
SMCREQ C1 PUT ENQ ID CTR IN QELSVRBA
D1 PUT QEL ADDR IN SVRB ESA
02 D4
08 A2
AUTOPRG A2 GET ADDR OF MOTHER OF TASK (TCBOTC)
B2 IS TCBOTC = ZERO?
YES
D2
NO
C2 IS MOTHER SCHEDULED FOR ABEND?
YES
NO
D2
GETFST D2 GET FIRST QEL ON QUEUE
E2 QUEUE EMPTY?
YES
NO
PRGRETA E3 SET UP REGISTER, SET ON SMCRBSW.
03 A4
F2 REPTEMIN DEQ ANY QEL ON QUEUE FOR THIS TASK (REG Y)
FINDTCB G2 REG X = TASK; REG Y = TASK'S DTR
TESTTCB H2 REG Y = ZERO?
NO
YES
J2 REG Y = TASK'S SISTER
K2 WERE WE AT TOP OF ABENDING TREE?
NO
YES
A4
K3 REG X = TASK'S MOTHER
A4
PRGRET A4 GET FIRST QEL
B4 ANY QEL'S ?
NO
YES
06 E2
SMCPROC A5 ENTRY ON SMC RESTART
B5 SET ON SMC SWITCH. SAVE REGISTERS 14 AND 15.
C5 SMC REG INDICATE SMC = STEP?
NO
04 F4
YES
04 E5

Chart 3-1 (page 9 of 25). ENQ/DEQ/RESERVE



IGC048
A1 --ENTER-- DEQUEUE
B1 ESTABLISH ADDRESSABILITY; SET UP POINTER TO PARAM LIST
C1 CHKLIST
CHECK VALIDITY OF PARAMETER LIST
D1 DIRECTED DEQ?
YES
NO
DIRDEQ
D2 GET TCB ADDR FROM PARM LIST -4. PUT IN TCB REG.
09 E1
PARMLOOP
E1 FINDMAJ
CHECK FOR EXISTING MAJOR QCB
F1 MAJOR QCB PRESENT ?
NO
YES
G1 GENERIC DEQ ?
NO
YES
NOGENDEQ
G2 FINDMIN
CHECK FOR EXISTING MINOR QCB
H1 SET ON GENERIC DEQ SWITCH. GET FIRST MINOR QCB
H2 MINOR QCB PRESENT ?
NO
YES
10 B2
09 H3
DQERR1
H3 RET = HAVE?
NO
YES
H4 -- ERROR -- BRANCH TO ERRX1
J1 SET RETURN CODE IN PARM LIST ELEMENT TO X'08'
J3 SET CODE OF X'08' IN RETURN CODE FIELD
11 C2
09 K1
TESTGMIN
K1 IS THERE A MINOR QCB
NO
YES
11 A5
10 A2

Chart 3-1 (page 10 of 25). ENQ/DEQ/RESERVE

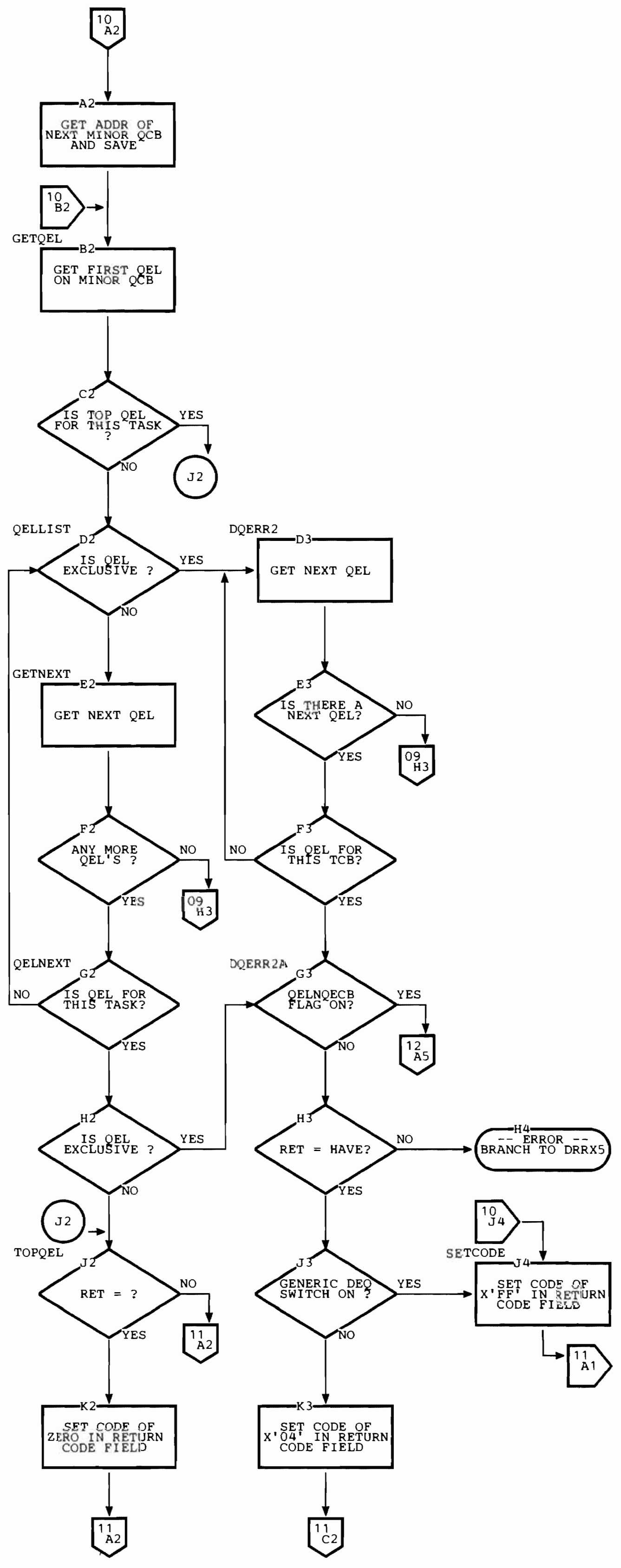


Chart 3-1 (page 11 of 25). ENQ/DEQ/RESERVE

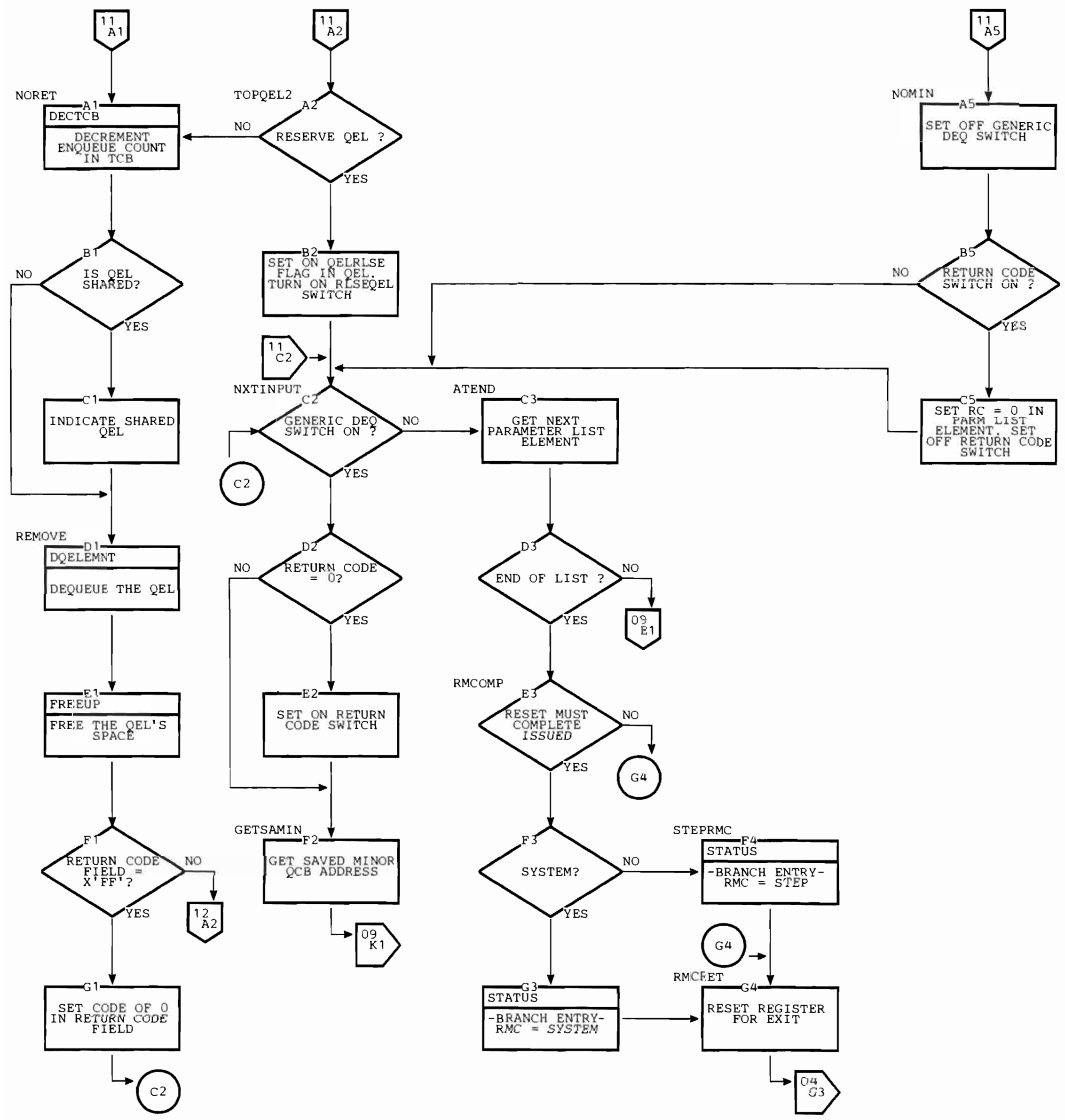
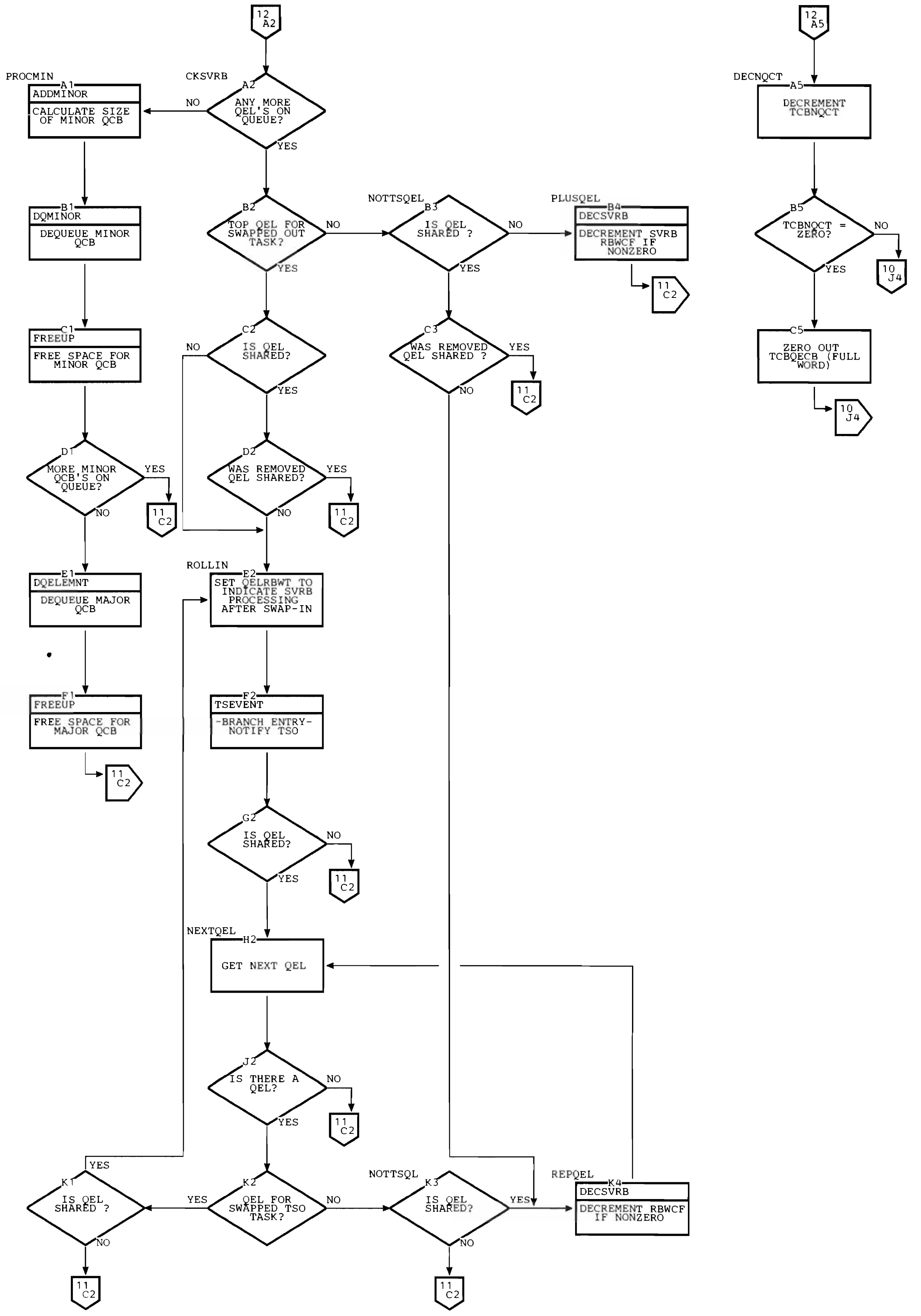
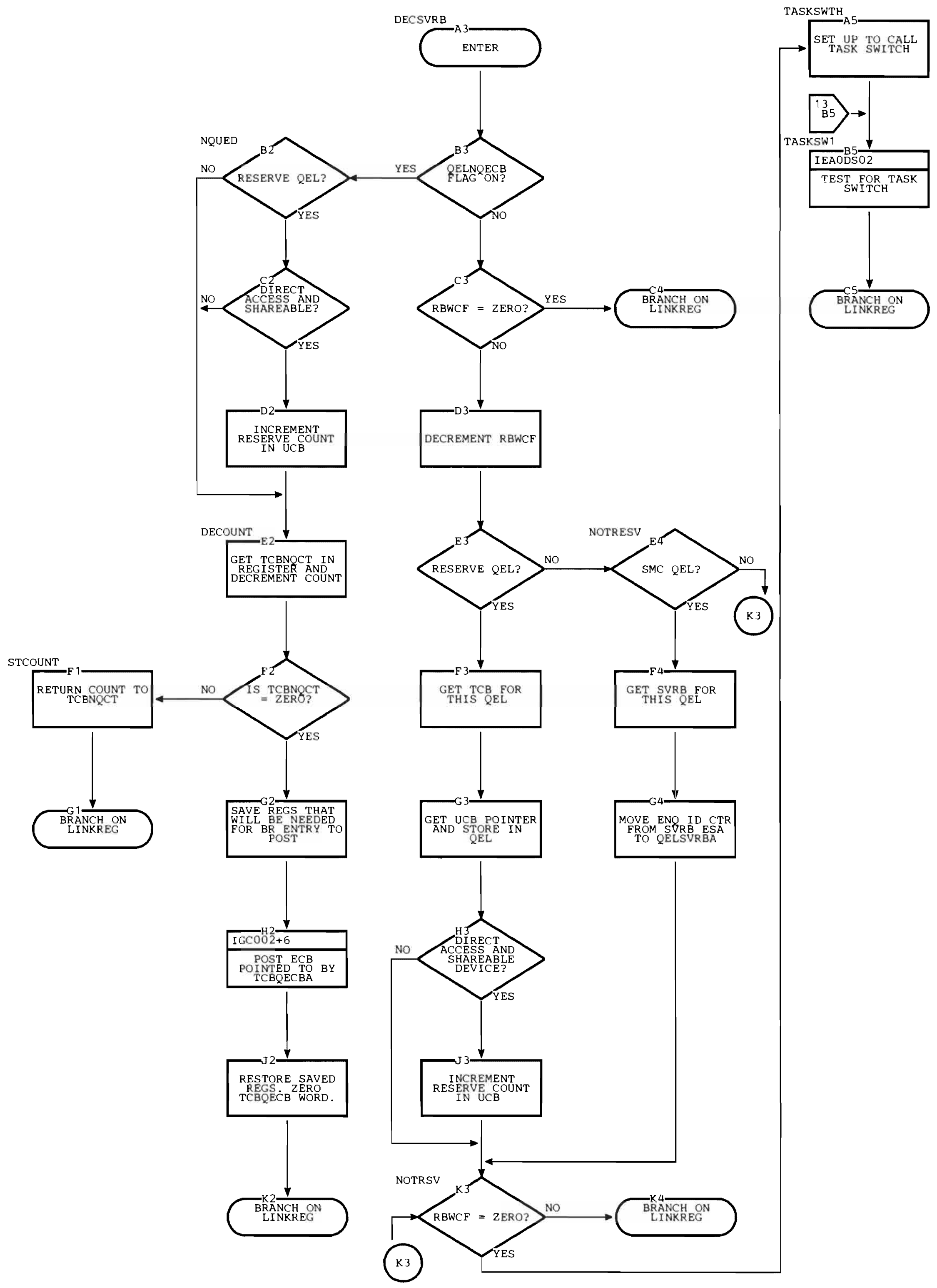


Chart 3-1 (page 12 of 25). ENQ/DEQ/RESERVE


12 A2
CKSVRB A2 ANY MORE QEL'S ON QUEUE?
NO
YES
PROCMIN A1 ADDMINOR CALCULATE SIZE OF MINOR QCB
B1 DQMINOR DEQUEUE MINOR QCB
C1 FREEUP FREE SPACE FOR MINOR QCB
D1 MORE MINOR QCB'S ON QUEUE?
YES 11 C2
NO
E1 DQELEMNT DEQUEUE MAJOR QCB
F1 FREEUP FREE SPACE FOR MAJOR QCB
11 C2
B2 TOP QEL FOR SWAPPED OUT TASK?
NO
YES
NOTTSQEL B3 IS QEL SHARED ?
NO
YES
PLUSQEL B4 DECSVRB DECREMENT SVRB RBWCF IF NONZERO
11 C2
C2 IS QEL SHARED?
NO
YES
D2 WAS REMOVED QEL SHARED?
YES 11 C2
NO
C3 WAS REMOVED QEL SHARED ?
YES 11 C2
NO
ROLLIN E2 SET QELRBWT TO INDICATE SVRB PROCESSING AFTER SWAP-IN
F2 TSEVENT -BRANCH ENTRY- NOTIFY TSO
G2 IS QEL SHARED?
NO 11 C2
YES
NEXTQEL H2 GET NEXT QEL
J2 IS THERE A QEL?
NO 11 C2
YES
K2 QEL FOR SWAPPED TSO TASK?
YES
NO
K1 IS QEL SHARED ?
YES
NO 11 C2
NOTTSQL K3 IS QEL SHARED?
YES
NO 11 C2
REPQEL K4 DECSVRB DECREMENT RBWCF IF NONZERO
12 A5
DECNQCT A5 DECREMENT TCBNQCT
B5 TCBNQCT = ZERO?
NO 10 J4
YES
C5 ZERO OUT TCBQECB (FULL WORD)
10 J4


DECSVRB
A3
ENTER
NQUED
B2
RESERVE QEL?
NO
YES
B3
QELNQECB FLAG ON?
YES
NO
C2
DIRECT ACCESS AND SHAREABLE?
NO
YES
C3
RBWCF = ZERO?
YES
NO
C4
BRANCH ON LINKREG
D2
INCREMENT RESERVE COUNT IN UCB
D3
DECREMENT RBWCF
DECOUNT
E2
GET TCBNQCT IN REGISTER AND DECREMENT COUNT
E3
RESERVE QEL?
NO
YES
NOTRESV
E4
SMC QEL?
NO
YES
K3
STCOUNT
F1
RETURN COUNT TO TCBNQCT
F2
IS TCBNQCT = ZERO?
NO
YES
F3
GET TCB FOR THIS QEL
F4
GET SVRB FOR THIS QEL
G1
BRANCH ON LINKREG
G2
SAVE REGS THAT WILL BE NEEDED FOR BR ENTRY TO POST
G3
GET UCB POINTER AND STORE IN QEL
G4
MOVE ENQ ID CTR FROM SVRB ESA TO QELSVRBA
H2
IGC002+6
POST ECB POINTED TO BY TCBQECBA
H3
DIRECT ACCESS AND SHAREABLE DEVICE?
NO
YES
J2
RESTORE SAVED REGS. ZERO TCBQECB WORD.
J3
INCREMENT RESERVE COUNT IN UCB
K2
BRANCH ON LINKREG
NOTRSV
K3
RBWCF = ZERO?
NO
YES
K3
K4
BRANCH ON LINKREG
TASKSWTH
A5
SET UP TO CALL TASK SWITCH
13 B5
TASKSW1
B5
IEA0DS02
TEST FOR TASK SWITCH
C5
BRANCH ON LINKREG

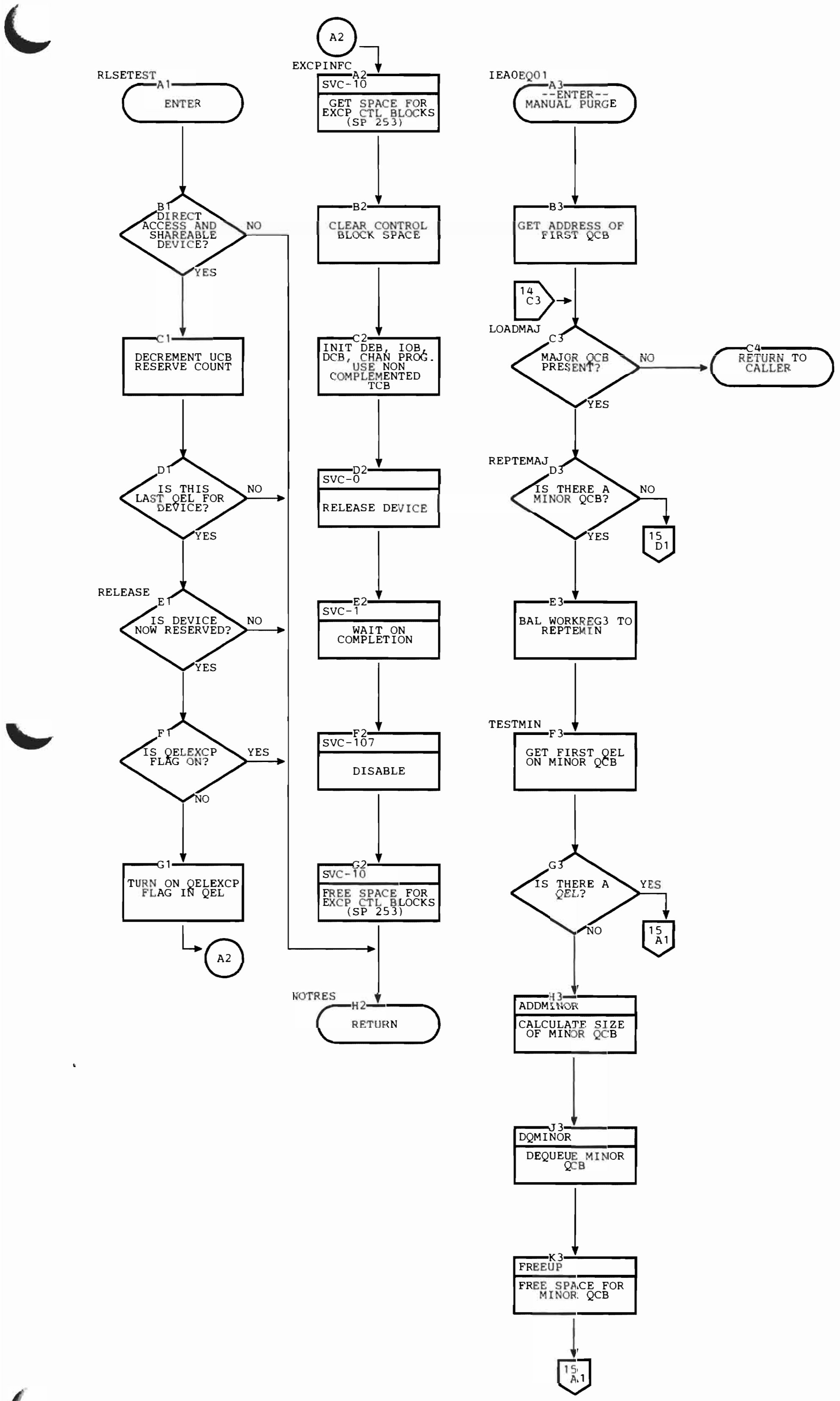
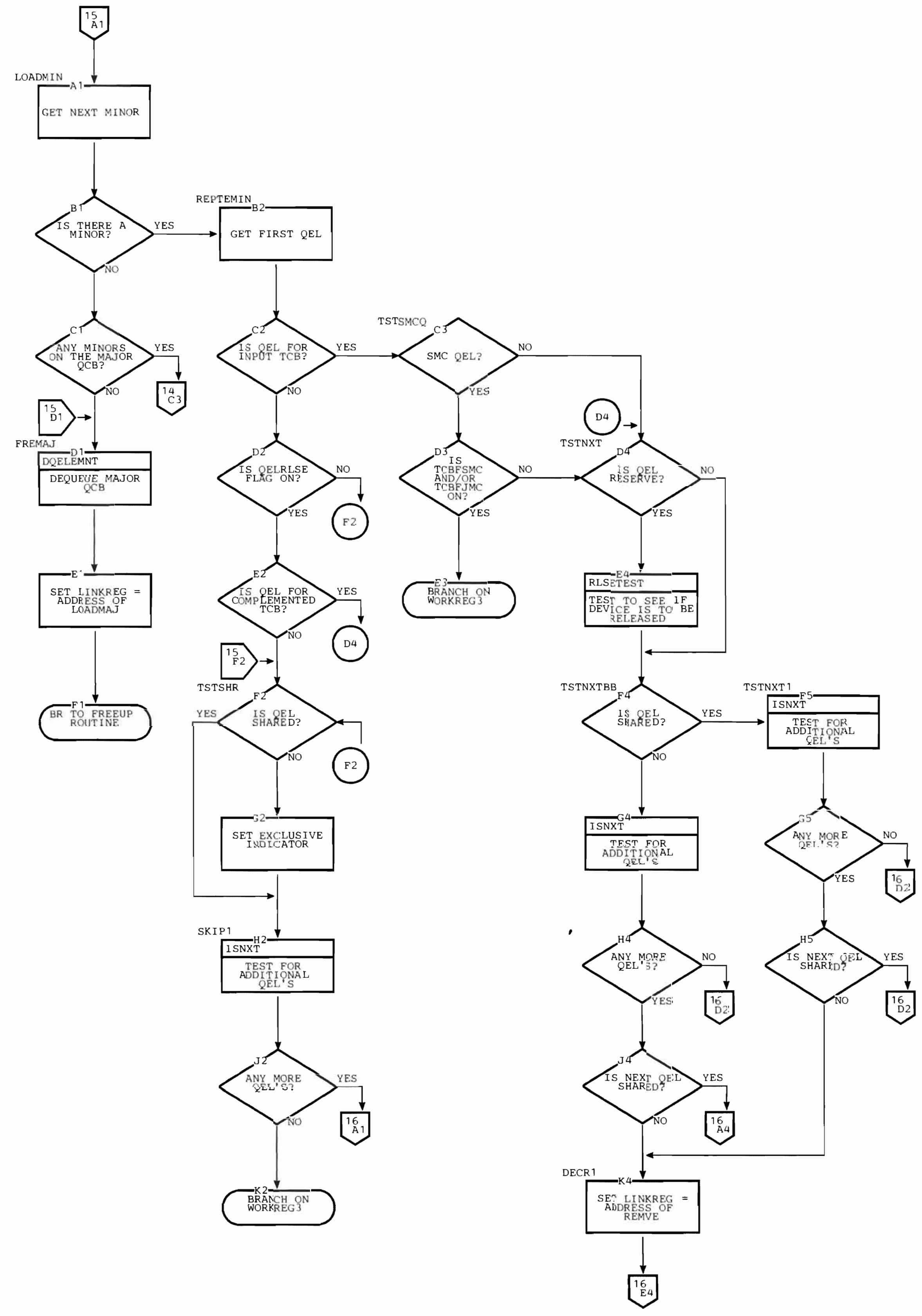

RLSETEST
A1
ENTER
B1
DIRECT ACCESS AND SHAREABLE DEVICE?
NO
YES
C1
DECREMENT UCB RESERVE COUNT
D1
IS THIS LAST QEL FOR DEVICE?
NO
YES
RELEASE
E1
IS DEVICE NOW RESERVED?
NO
YES
F1
IS QELEXCP FLAG ON?
YES
NO
G1
TURN ON QELEXCP FLAG IN QEL
A2
A2
EXCPINFC
A2
SVC-10
GET SPACE FOR EXCP CTL BLOCKS (SP 253)
B2
CLEAR CONTROL BLOCK SPACE
C2
INIT DEB, IOB, DCB, CHAN PROG. USE NON COMPLEMENTED TCB
D2
SVC-0
RELEASE DEVICE
E2
SVC-1
WAIT ON COMPLETION
F2
SVC-107
DISABLE
G2
SVC-10
FREE SPACE FOR EXCP CTL BLOCKS (SP 253)
NOTRES
H2
RETURN
IEAOEQ01
A3
--ENTER-- MANUAL PURGE
B3
GET ADDRESS OF FIRST QCB
14 C3
LOADMAJ
C3
MAJOR QCB PRESENT?
NO
YES
C4
RETURN TO CALLER
REPTEMAJ
D3
IS THERE A MINOR QCB?
NO
YES
15 D1
E3
BAL WORKREG3 TO REPTEMIN
TESTMIN
F3
GET FIRST QEL ON MINOR QCB
G3
IS THERE A QEL?
YES
NO
15 A1
H3
ADDMINOR
CALCULATE SIZE OF MINOR QCB
J3
DQMINOR
DEQUEUE MINOR QCB
K3
FREEUP
FREE SPACE FOR MINOR QCB
15 A1

Chart 3-1 (page 15 of 25). ENQ/DEQ/RESERVE



15 A1
LOADMIN A1 GET NEXT MINOR
B1 IS THERE A MINOR?
YES
NO
REPTEMIN B2 GET FIRST QEL
C1 ANY MINORS ON THE MAJOR QCB?
YES
14 C3
NO
15 D1
FREMAJ D1 DQELEMNT DEQUEUE MAJOR QCB
E1 SET LINKREG = ADDRESS OF LOADMAJ
F1 BR TO FREEUP ROUTINE
C2 IS QEL FOR INPUT TCB?
YES
NO
TSTSMCQ C3 SMC QEL?
NO
YES
D2 IS QELRLSE FLAG ON?
NO
F2
YES
E2 IS QEL FOR COMPLEMENTED TCB?
YES
D4
NO
15 F2
TSTSHR F2 IS QEL SHARED?
YES
NO
F2
G2 SET EXCLUSIVE INDICATOR
SKIP1 H2 ISNXT TEST FOR ADDITIONAL QEL'S
J2 ANY MORE QEL'S?
YES
16 A1
NO
K2 BRANCH ON WORKREG3
D3 IS TCBFSMC AND/OR TCBFJMC ON?
NO
YES
E3 BRANCH ON WORKREG3
D4
TSTNXT D4 IS QEL RESERVE?
NO
YES
E4 RLSETEST TEST TO SEE IF DEVICE IS TO BE RELEASED
TSTNXTBB F4 IS QEL SHARED?
YES
NO
TSTNXT1 F5 ISNXT TEST FOR ADDITIONAL QEL'S
G4 ISNXT TEST FOR ADDITIONAL QEL'S
G5 ANY MORE QEL'S?
NO
16 D2
YES
H4 ANY MORE QEL'S?
NO
16 D2
YES
H5 IS NEXT QEL SHARED?
YES
16 D2
NO
J4 IS NEXT QEL SHARED?
YES
16 A4
NO
DECR1 K4 SET LINKREG = ADDRESS OF REMVE
16 E4

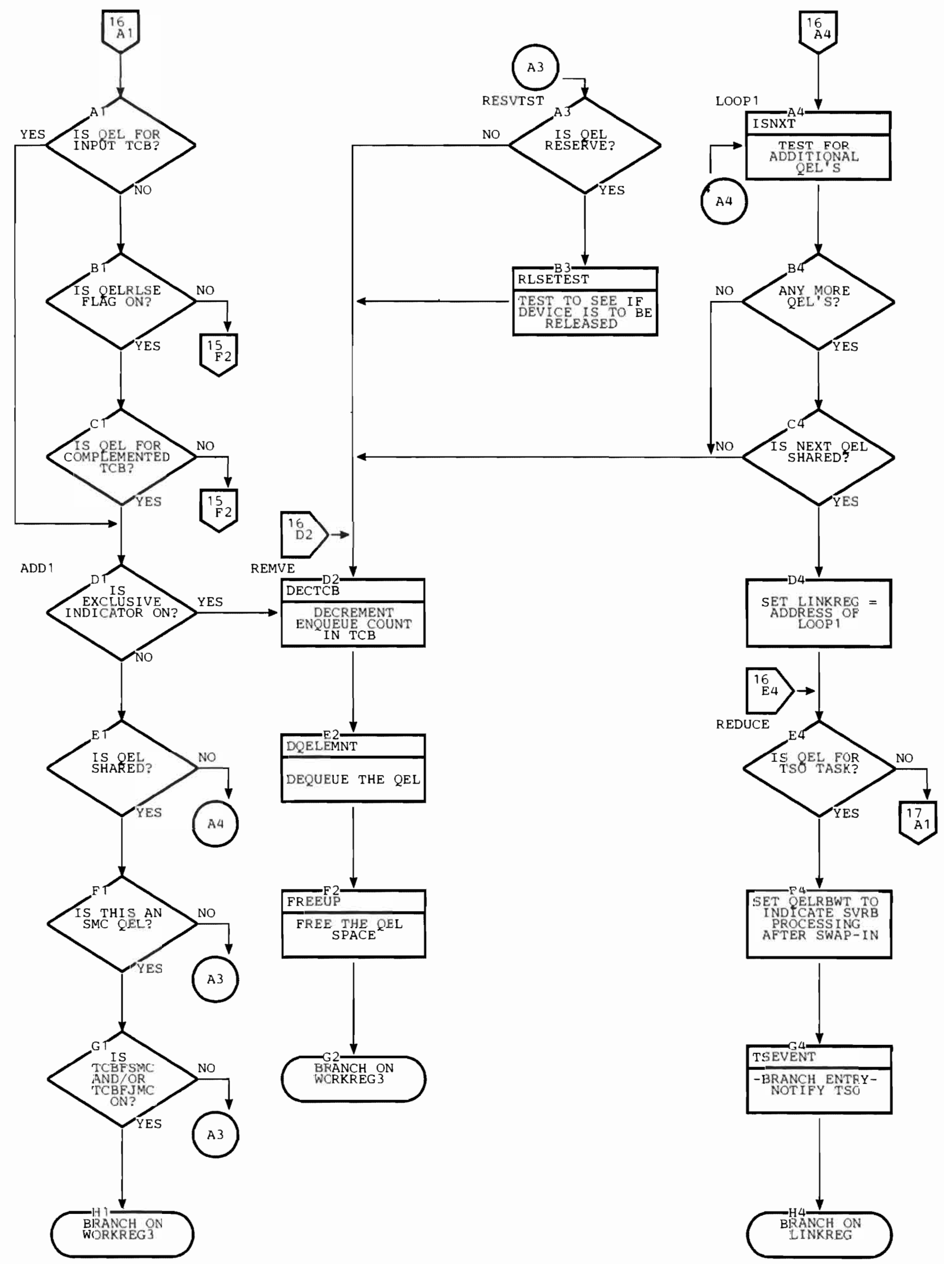

16 A1
A1 IS QEL FOR INPUT TCB?
YES
NO
B1 IS QELRLSE FLAG ON?
NO
15 F2
YES
C1 IS QEL FOR COMPLEMENTED TCB?
NO
15 F2
YES
ADD1
D1 IS EXCLUSIVE INDICATOR ON?
YES
NO
E1 IS QEL SHARED?
NO
A4
YES
F1 IS THIS AN SMC QEL?
NO
A3
YES
G1 IS TCBFSMC AND/OR TCBFJMC ON?
NO
A3
YES
H1 BRANCH ON WORKREG3
A3
RESVTST
A3 IS QEL RESERVE?
NO
YES
B3 RLSETEST
TEST TO SEE IF DEVICE IS TO BE RELEASED
16 D2
REMVE
D2 DECTCB
DECREMENT ENQUEUE COUNT IN TCB
E2 DQELEMNT
DEQUEUE THE QEL
F2 FREEUP
FREE THE QEL SPACE
G2 BRANCH ON WORKREG3
16 A4
LOOP1
A4 ISNXT
TEST FOR ADDITIONAL QEL'S
A4
B4 ANY MORE QEL'S?
NO
YES
C4 IS NEXT QEL SHARED?
NO
YES
D4 SET LINKREG = ADDRESS OF LOOP1
16 E4
REDUCE
E4 IS QEL FOR TSO TASK?
NO
17 A1
YES
F4 SET QELRBWT TO INDICATE SVRB PROCESSING AFTER SWAP-IN
G4 TSEVENT
-BRANCH ENTRY-NOTIFY TSO
H4 BRANCH ON LINKREG

Chart 3-1 (page 17 of 25). ENQ/DEQ/RESERVE

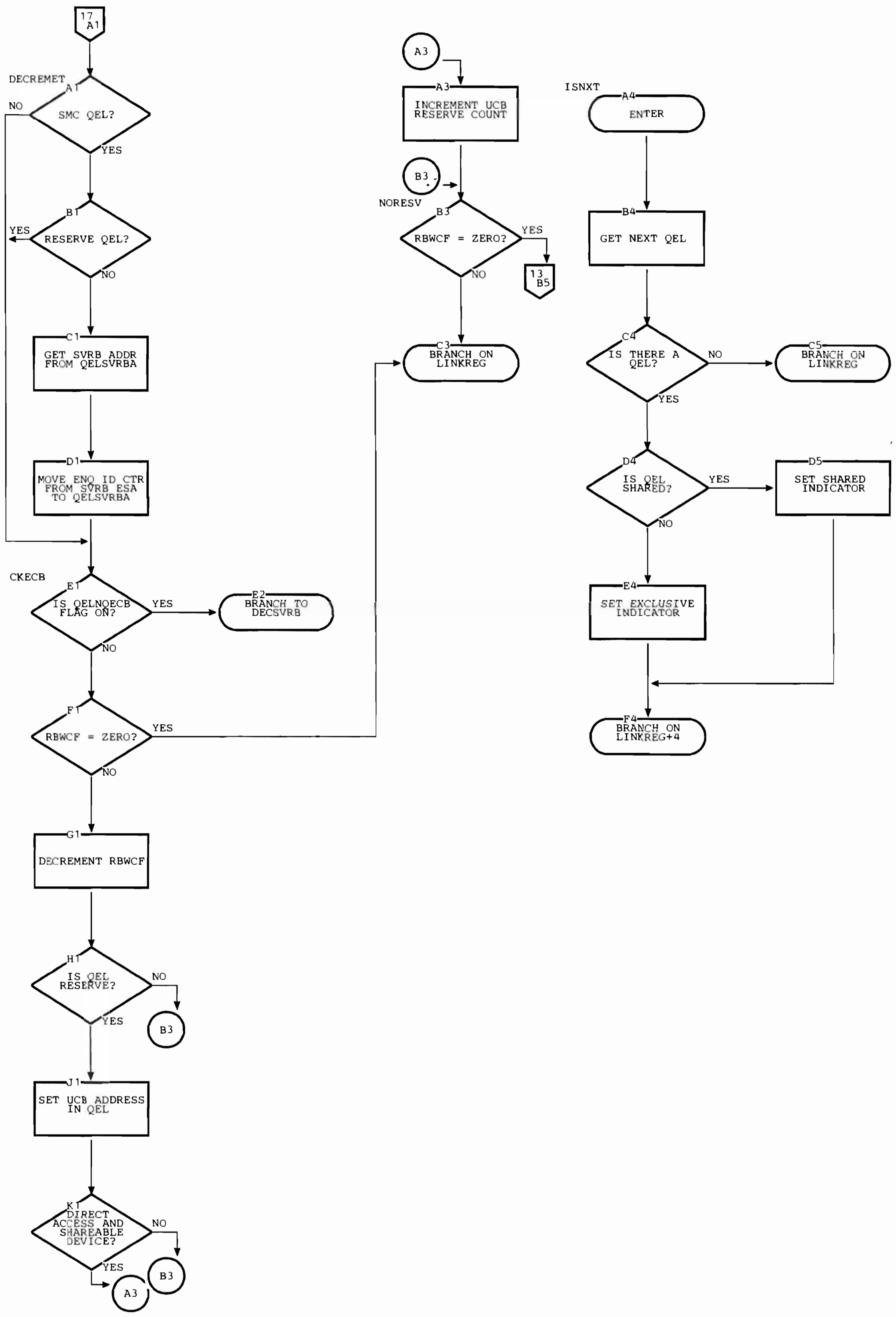
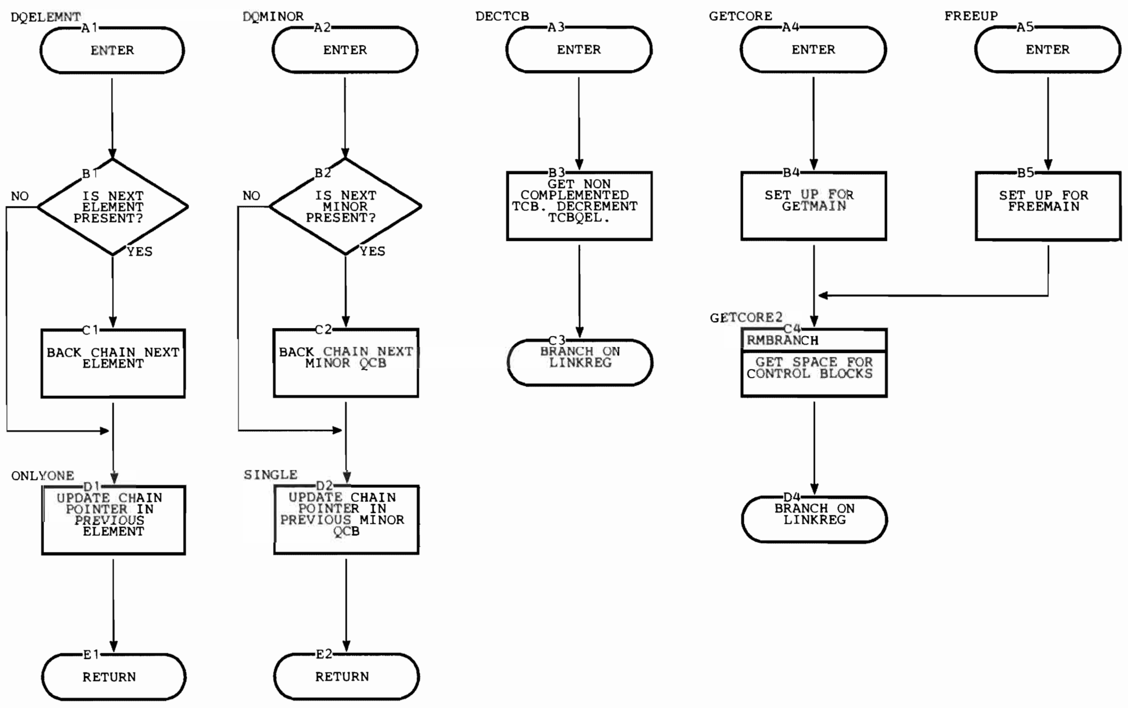


Chart 3-1 (page 18 of 25). ENQ/DEQ/RESERVE


DQELEMNT
A1 ENTER
B1 IS NEXT ELEMENT PRESENT?
NO
YES
C1 BACK CHAIN NEXT ELEMENT
ONLYONE
D1 UPDATE CHAIN POINTER IN PREVIOUS ELEMENT
E1 RETURN
DQMINOR
A2 ENTER
B2 IS NEXT MINOR PRESENT?
NO
YES
C2 BACK CHAIN NEXT MINOR QCB
SINGLE
D2 UPDATE CHAIN POINTER IN PREVIOUS MINOR QCB
E2 RETURN
DECTCB
A3 ENTER
B3 GET NON COMPLEMENTED TCB. DECREMENT TCBQEL.
C3 BRANCH ON LINKREG
GETCORE
A4 ENTER
B4 SET UP FOR GETMAIN
GETCORE2
C4 RMBRANCH
GET SPACE FOR CONTROL BLOCKS
D4 BRANCH ON LINKREG
FREEUP
A5 ENTER
B5 SET UP FOR FREEMAIN

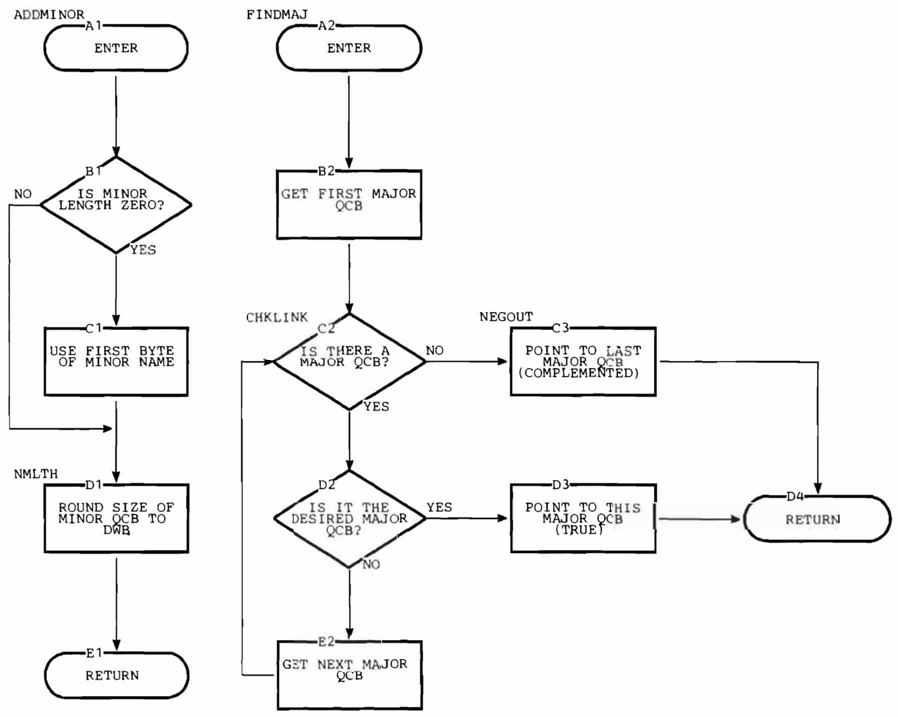
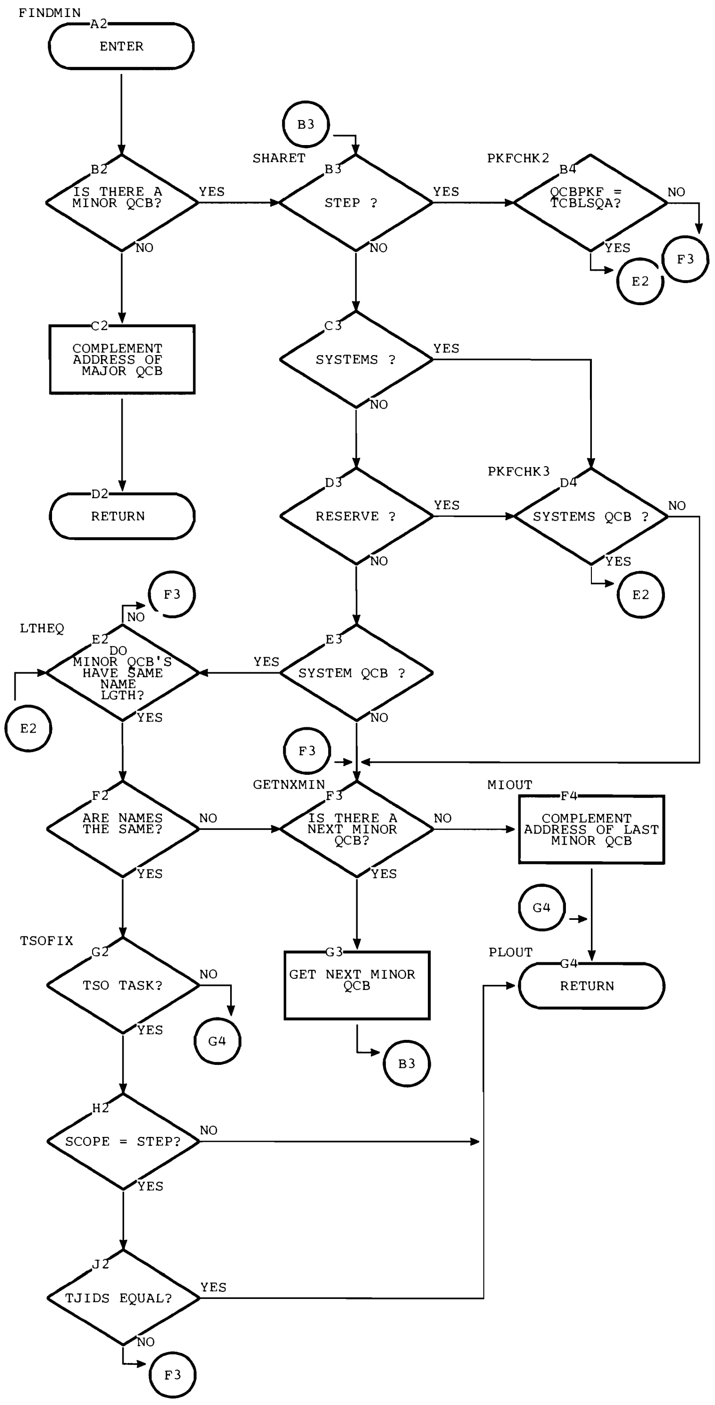
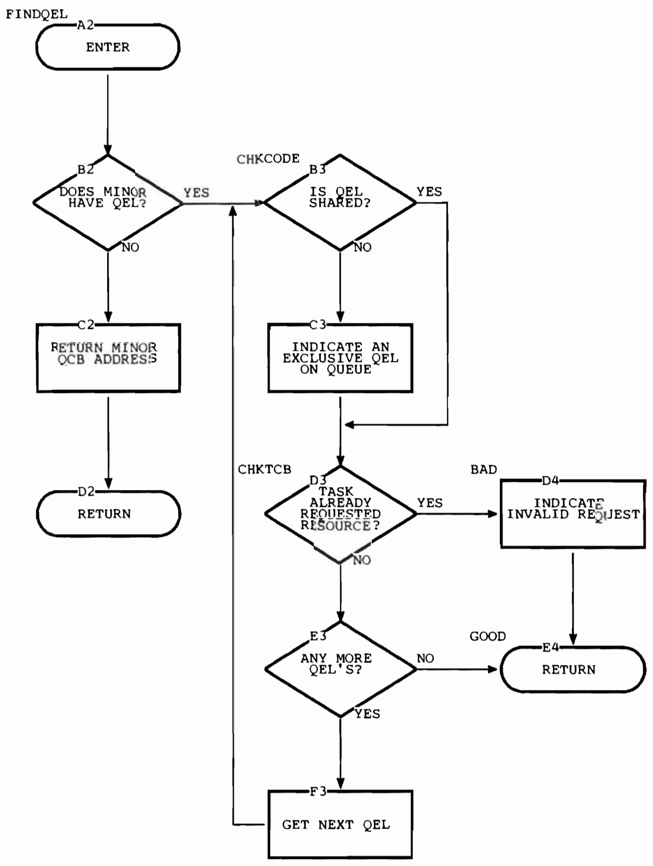

ADDMINOR
A1
ENTER
B1
IS MINOR LENGTH ZERO?
NO
YES
C1
USE FIRST BYTE OF MINOR NAME
NMLTH
D1
ROUND SIZE OF MINOR QCB TO DWB
E1
RETURN
FINDMAJ
A2
ENTER
B2
GET FIRST MAJOR QCB
CHKLINK
C2
IS THERE A MAJOR QCB?
NO
YES
D2
IS IT THE DESIRED MAJOR QCB?
YES
NO
E2
GET NEXT MAJOR QCB
NEGOUT
C3
POINT TO LAST MAJOR QCB (COMPLEMENTED)
D3
POINT TO THIS MAJOR QCB (TRUE)
D4
RETURN

Chart 3-1 (page 20 of 25). ENQ/DEQ/RESERVE




FINDQEL
A2
ENTER
B2
DOES MINOR HAVE QEL?
YES
NO
C2
RETURN MINOR QCB ADDRESS
D2
RETURN
CHKCODE
B3
IS QEL SHARED?
YES
NO
C3
INDICATE AN EXCLUSIVE QEL ON QUEUE
CHKTCB
D3
TASK ALREADY REQUESTED RESOURCE?
YES
NO
BAD
D4
INDICATE INVALID REQUEST
E3
ANY MORE QEL'S?
NO
YES
GOOD
E4
RETURN
F3
GET NEXT QEL

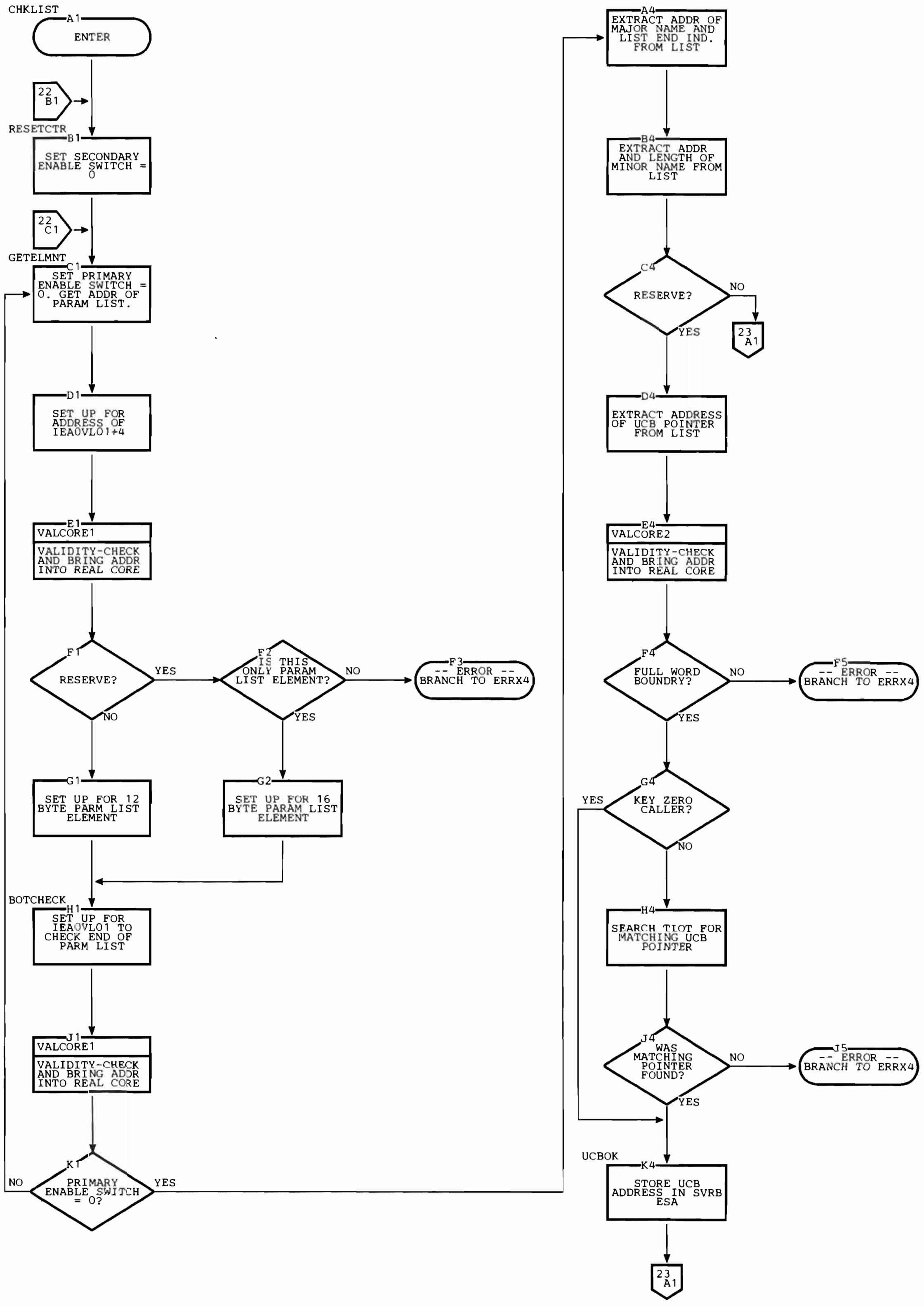

CHKLIST
A1
ENTER
22 B1
RESETCTR
B1
SET SECONDARY ENABLE SWITCH = 0
22 C1
GETELMNT
C1
SET PRIMARY ENABLE SWITCH = 0. GET ADDR OF PARAM LIST.
D1
SET UP FOR ADDRESS OF IEA0VL01+4
E1
VALCORE1
VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
F1
RESERVE?
YES
NO
F2
IS THIS ONLY PARAM LIST ELEMENT?
NO
YES
F3
-- ERROR -- BRANCH TO ERRX4
G1
SET UP FOR 12 BYTE PARM LIST ELEMENT
G2
SET UP FOR 16 BYTE PARAM LIST ELEMENT
BOTCHECK
H1
SET UP FOR IEA0VL01 TO CHECK END OF PARM LIST
J1
VALCORE1
VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
K1
PRIMARY ENABLE SWITCH = 0?
NO
YES
A4
EXTRACT ADDR OF MAJOR NAME AND LIST END IND. FROM LIST
B4
EXTRACT ADDR AND LENGTH OF MINOR NAME FROM LIST
C4
RESERVE?
NO
YES
23 A1
D4
EXTRACT ADDRESS OF UCB POINTER FROM LIST
E4
VALCORE2
VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
F4
FULL WORD BOUNDRY?
NO
YES
F5
-- ERROR -- BRANCH TO ERRX4
G4
KEY ZERO CALLER?
YES
NO
H4
SEARCH TIOT FOR MATCHING UCB POINTER
J4
WAS MATCHING POINTER FOUND?
NO
YES
J5
-- ERROR -- BRANCH TO ERRX4
UCBOK
K4
STORE UCB ADDRESS IN SVRB ESA
23 A1

Chart 3-1 (page 23 of 25). ENQ/DEQ/RESERVE

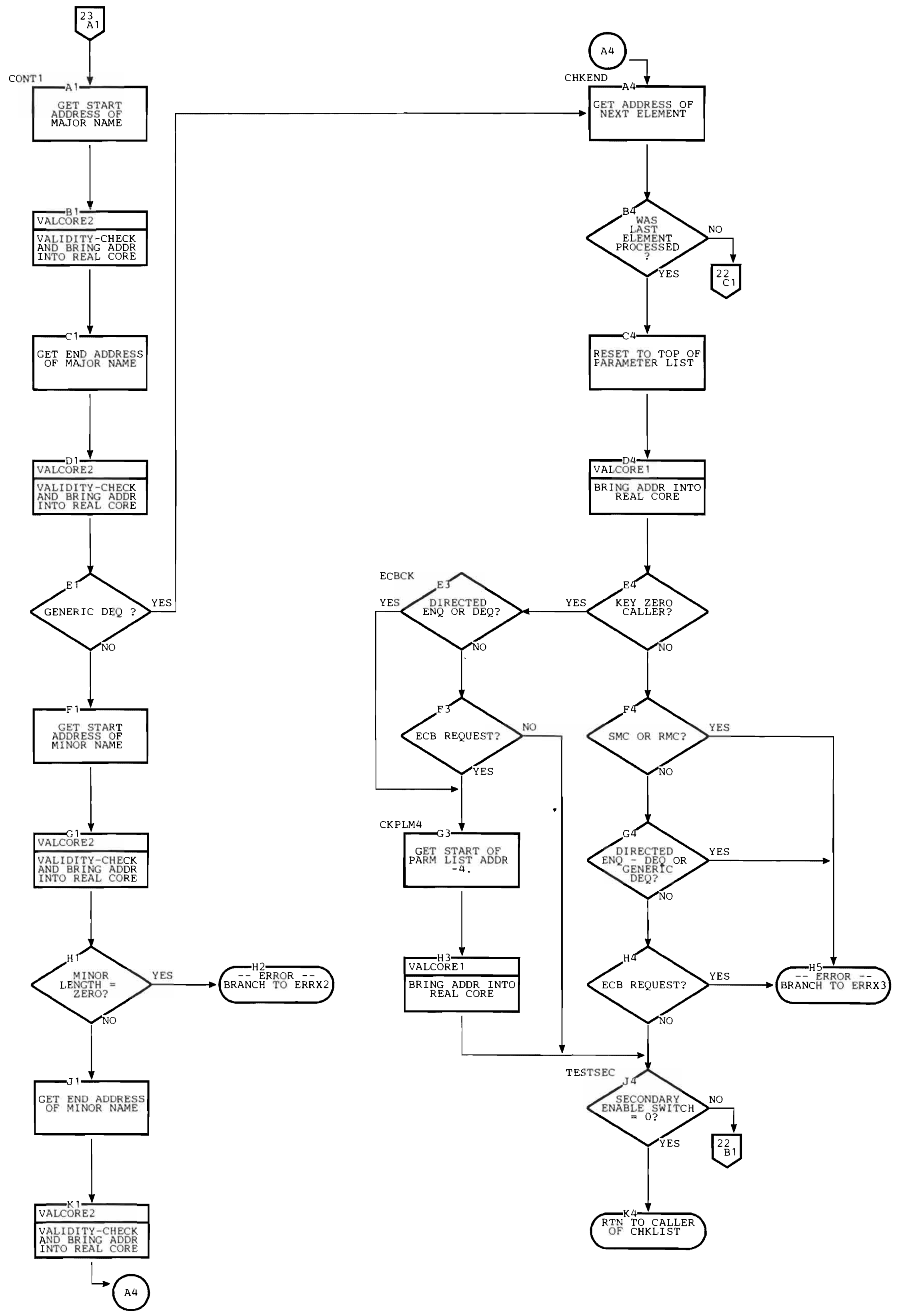

23 A1
CONT1
A1 GET START ADDRESS OF MAJOR NAME
B1 VALCORE2 VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
C1 GET END ADDRESS OF MAJOR NAME
D1 VALCORE2 VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
E1 GENERIC DEQ ?
YES
NO
F1 GET START ADDRESS OF MINOR NAME
G1 VALCORE2 VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
H1 MINOR LENGTH = ZERO?
YES
H2 -- ERROR -- BRANCH TO ERRX2
NO
J1 GET END ADDRESS OF MINOR NAME
K1 VALCORE2 VALIDITY-CHECK AND BRING ADDR INTO REAL CORE
A4
A4
CHKEND
A4 GET ADDRESS OF NEXT ELEMENT
B4 WAS LAST ELEMENT PROCESSED ?
NO
22 C1
YES
C4 RESET TO TOP OF PARAMETER LIST
D4 VALCORE1 BRING ADDR INTO REAL CORE
E4 KEY ZERO CALLER?
YES
NO
ECBCK
E3 DIRECTED ENQ OR DEQ?
YES
NO
F3 ECB REQUEST?
NO
YES
CKPLM4
G3 GET START OF PARM LIST ADDR -4.
H3 VALCORE1 BRING ADDR INTO REAL CORE
F4 SMC OR RMC?
YES
NO
G4 DIRECTED ENQ - DEQ OR GENERIC DEQ?
YES
NO
H4 ECB REQUEST?
YES
H5 -- ERROR -- BRANCH TO ERRX3
NO
TESTSEC
J4 SECONDARY ENABLE SWITCH = 0?
NO
22 B1
YES
K4 RTN TO CALLER OF CHKLIST

Chart 3-1 (page 24 of 25). ENQ/DEQ/RESERVE

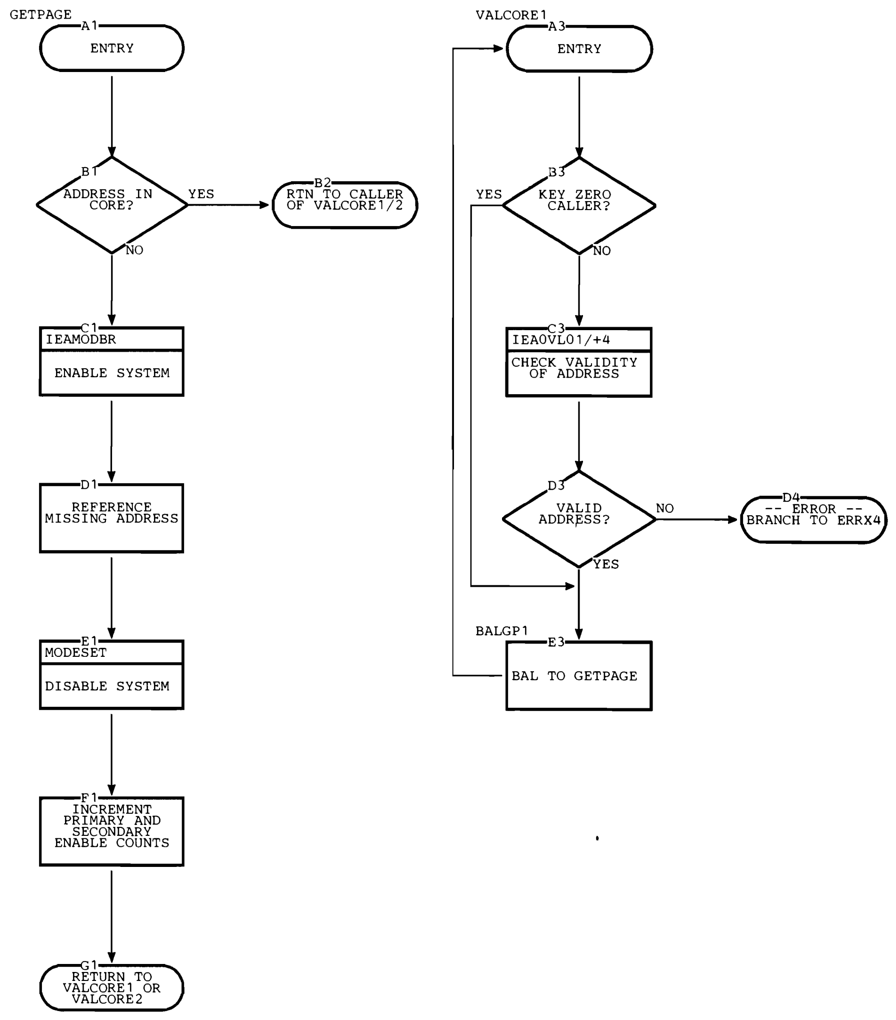
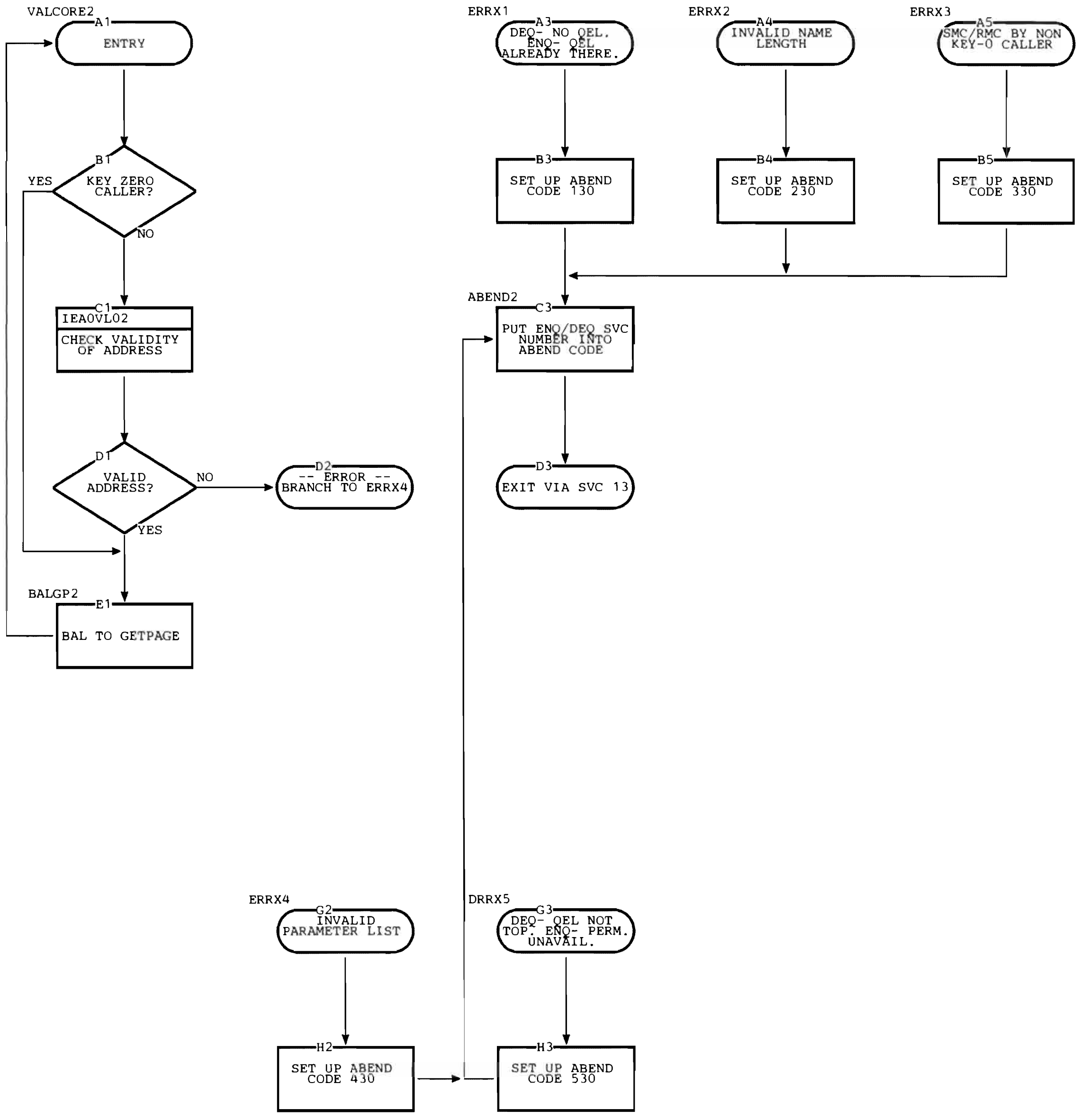


Chart 3-1 (page 25 of 25). ENQ/DEQ/RESERVE



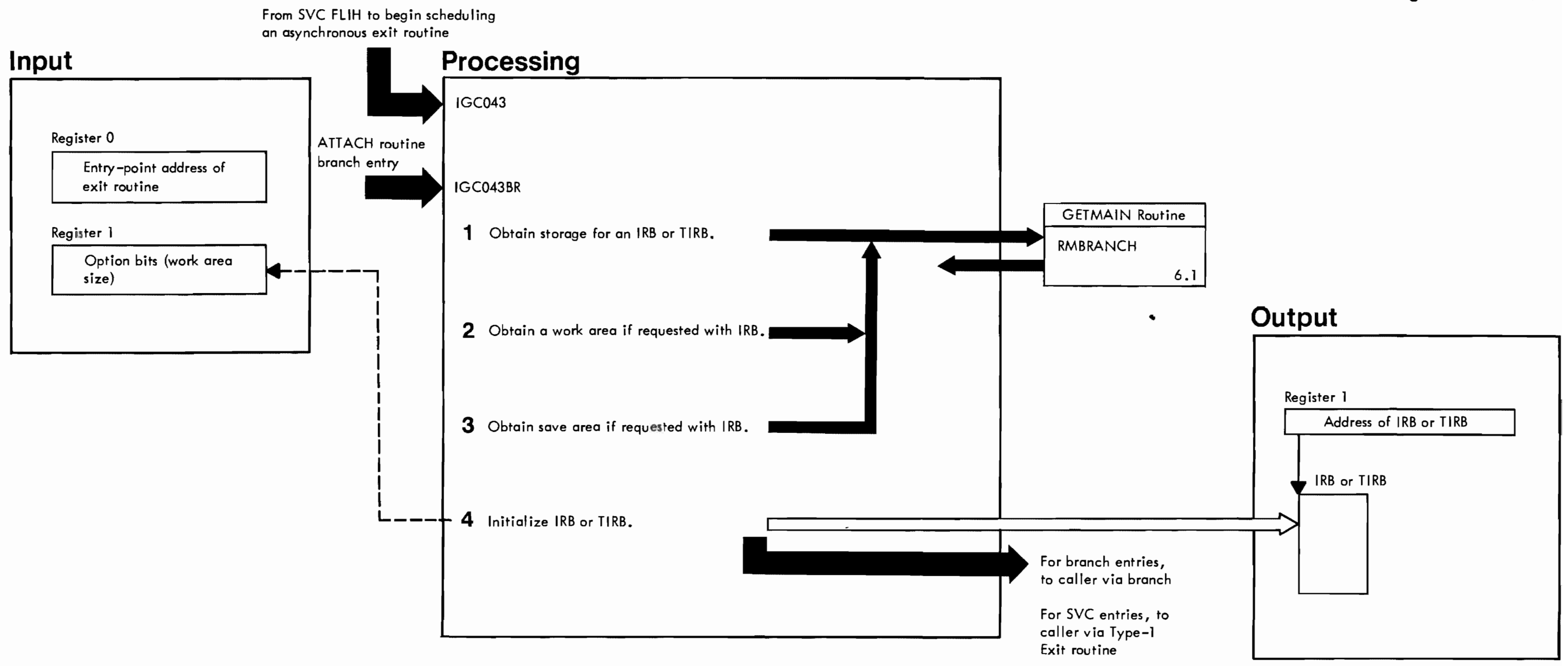

From SVC FLIH to begin scheduling an asynchronous exit routine
Input
Register 0
Entry-point address of exit routine
Register 1
Option bits (work area size)
ATTACH routine branch entry
Processing
IGC043
IGC043BR
1 Obtain storage for an IRB or TIRB.
2 Obtain a work area if requested with IRB.
3 Obtain save area if requested with IRB.
4 Initialize IRB or TIRB.
GETMAIN Routine
RMBRANCH
6.1
For branch entries, to caller via branch
For SVC entries, to caller via Type-1 Exit routine
Output
Register 1
Address of IRB or TIRB
IRB or TIRB

Diagram 3.11 Stage 1 Exit Effector (Module IEAVEF00)

NOTES	ROUTINE NAME	LABEL
A user program may request the future execution of an exit routine to handle an unpredictable 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).	Stage 1 Exit	IGC043
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. Note that these routines do not schedule the execution of user program-check routines. ABEND processing that results from a program interruption can be intercepted by a SPIE macro instruction or, in the absence of SPIE, by a STAE macro instruction. See Diagram 3.5 for a description of the SPIE routine.		
Through the exit effectors, a supervisor routine can also request that a supervisor service be deferred until the supervisor routine can operate under the TCB of the task it is servicing. This prevents page faults at times when system integrity or performance might be impaired.		
The Stage 1 Exit Effector is called by supervisor or data management routines. Its purpose is to create and initialize, according to input parameters, either an IRB (interruption request block) to control a user exit routine whose future use is requested by the caller, or a TIRB (task interruption request block) that permits the supervisor to defer execution until it can execute under the TCB of the task being serviced.		
Data management routines enter the Stage 1 Exit Effector at IGC043 to begin scheduling routines that handle asynchronous events.		
The ATTACH routine enters at IGC043BR to create an IRB and a TIRB for a new subtask.		
2 The caller uses the work area to build IQEs (interruption queue elements). For example, the ATTACH routine builds an IQE for each asynchronous exit routine.		
3 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, and the PSW to be loaded to start execution of the asynchronous exit routine.		

Diagram 3.12
Stage 2 Exit Effector

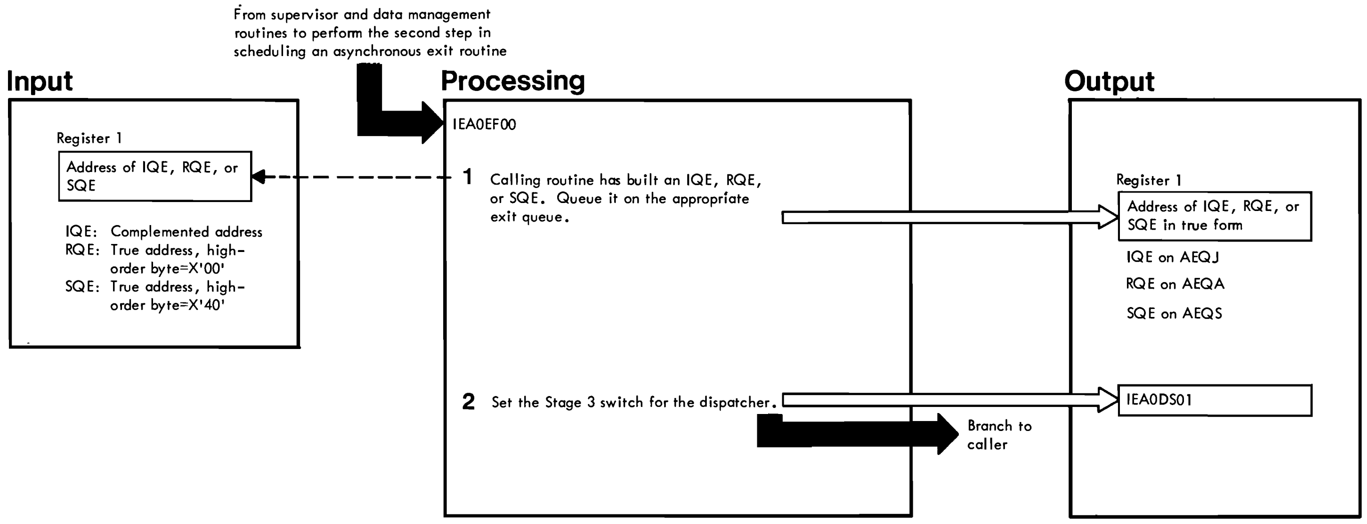
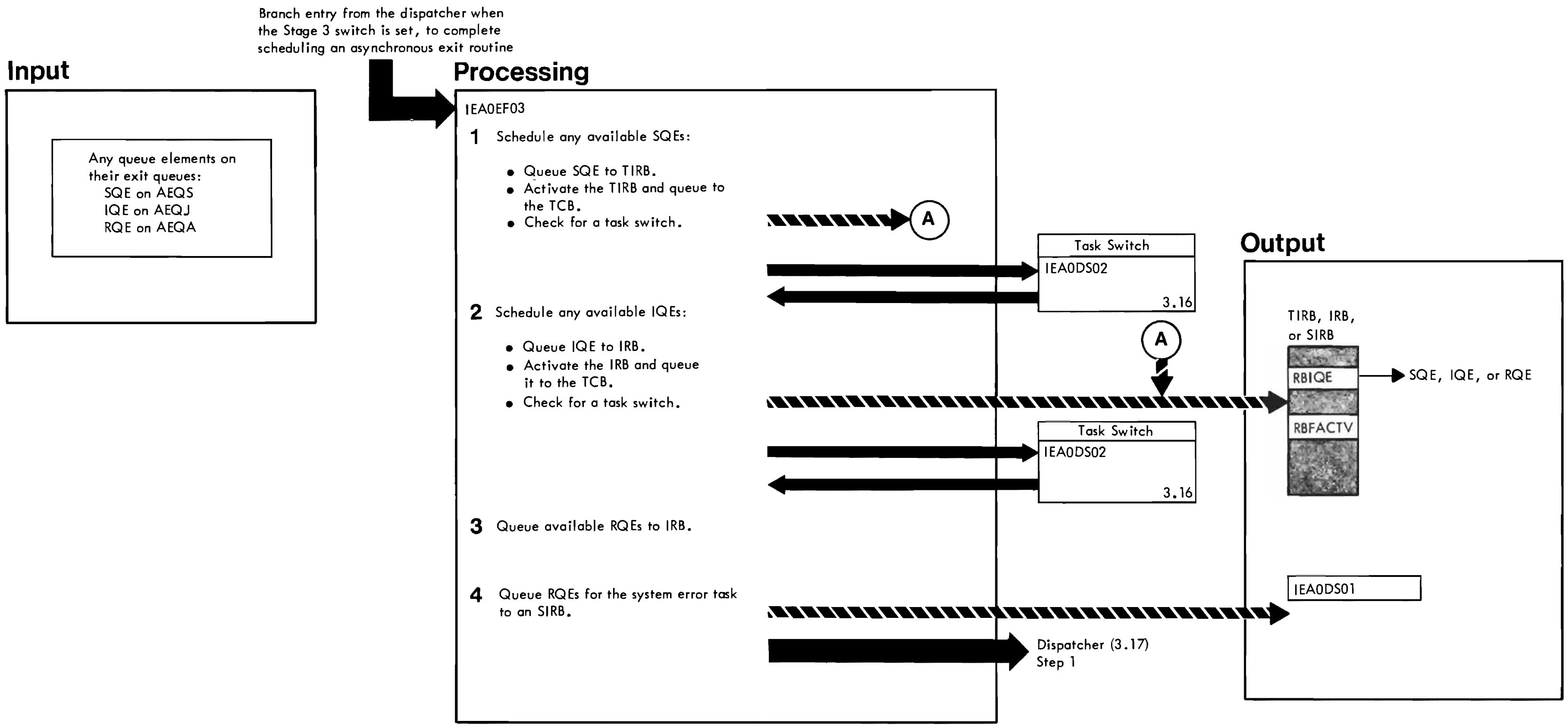

From supervisor and data management routines to perform the second step in scheduling an asynchronous exit routine
Input
Register 1
Address of IQE, RQE, or SQE
IQE: Complemented address
RQE: True address, high-order byte=X'00'
SQE: True address, high-order byte=X'40'
Processing
IEA0EF00
1 Calling routine has built an IQE, RQE, or SQE. Queue it on the appropriate exit queue.
2 Set the Stage 3 switch for the dispatcher.
Branch to caller
Output
Register 1
Address of IQE, RQE, or SQE in true form
IQE on AEQJ
RQE on AEQA
SQE on AEQS
IEA0DS01

Diagram 3.12 Stage 2 Exit Effector (Module IEAVNU00)

NOTES	ROUTINE NAME	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 SQE (supervisor queue element).	Stage 2 Exit Effector	IEAEF00

Type of Queue <u>Element</u>	<u>Purpose</u>	Type of Exit <u>Queue</u>
IQE	Supervisor routine wants to schedule an asynchronous exit routine.	AEQJ
RQE	Data management routine wants to schedule an asynchronous routine.	AEQA
SQE	Supervisor wants to schedule an asynchronous supervisor service.	AEQS

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.



Branch entry from the dispatcher when the Stage 3 switch is set, to complete scheduling an asynchronous exit routine

Input

Any queue elements on their exit queues:
SQE on AEQS
IQE on AEQJ
RQE on AEQA

Processing

IEAOEF03

1 Schedule any available SQEs:

- Queue SQE to TIRB.
- Activate the TIRB and queue to the TCB.
- Check for a task switch.

2 Schedule any available IQEs:

- Queue IQE to IRB.
- Activate the IRB and queue it to the TCB.
- Check for a task switch.

3 Queue available RQEs to IRB.

4 Queue RQEs for the system error task to an SIRB.

Output

Diagram 3.13 (Steps 1-4) Stage 3 Exit Effector (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
The dispatcher enters the Stage 3 Exit Effector when it finds that Stage 2 has placed at least one element on an asynchronous exit queue and turned on IEA0DS01 to signal that an asynchronous exit routine or asynchronous event is ready for scheduling.	Stage 3 Exit Effector	IEA0EF03
1 SQEs represent a request for a supervisor service on an asynchronous basis. An SQE cannot be processed if: • The TCB that the SQE points to represents a task responsible for setting the supervisor lock (set if a disabled page fault occurs). • The SQE is in purge status (Bit 0 of SQEFLAGS=1) ABTERM uses the SQE/TIRB processing to schedule ABEND. If the TCB is being abnormally terminated, only the special ABTERM SQE will be scheduled.		
2 An IQE cannot be processed if: • Asynchronous events are not allowed for the task requesting the exit routine. • A TIRB is at the top of the TCB's RB chain. A task can become interlocked if it depends on asynchronous processing represented by a TIRB occuring before processing represented by an IRB. • The TCB that the IQE represents is the task responsible for setting the supervisor lock. If an IQE exists, the Stage 3 Exit Effector determines whether its IRB is a normal IRB or an attention IRB for a TSO task. If the IRB is not an attention IRB (RBATNXIT=1), execution continues with the chaining process. If it is an attention IRB, the TCB associated with this IRB and all lower-level tasks in this user's task structure are checked to determine whether attention exits and asynchronous exits are permitted. If not, processing continues with the next IQE. If these exits are permitted, the IQE is scheduled by queueing the IQE in the usual manner.		IRBINTL

NOTES	ROUTINE NAME	LABEL
3 The same limitations that apply to IQEs apply to RQEs.	Stage 3 Exit Effector	
4 One or more of the RQEs may represent a request by an I/O routine for the use of a system error handling routine. For each RQE representing a request to use an error routine, Stage 3 tests first whether a special system request block is "active," that is, already queued to its system error TCB. The system error TCB is a permanent TCB of high priority whose current "dummy" RB is normally in a wait condition. The request block that represents a system error handling routine is called the system interruption request block, or SIRB. If the SIRB is already queued to the system error TCB, the "active" bit (RBFACTV) is set in the SIRB's status field. In this case, the error routine has already been scheduled for another request. The new request must then be deferred and await the next execution of Stage 3, when the dispatcher is next entered. If the SIRB is not "active," that is, not queued to the system error TCB, Stage 3 removes the RQE from its asynchronous exit queue and queues it to the SIRB. Stage 3 then initializes the SIRB. As part of this initialization, it sets the RB old PSW to provide reentry to the "error fetch sequence" (ERFETCH) of Stage 3. Besides altering the RB old PSW, Stage 3 queues the SIRB to the system error TCB. After Stage 3 queues the SIRB to the system error TCB, it defers subsequent error requests on the RQE queue. These requests are not processed until the SIRB becomes "inactive," removed from its TCB during execution of the routine.		

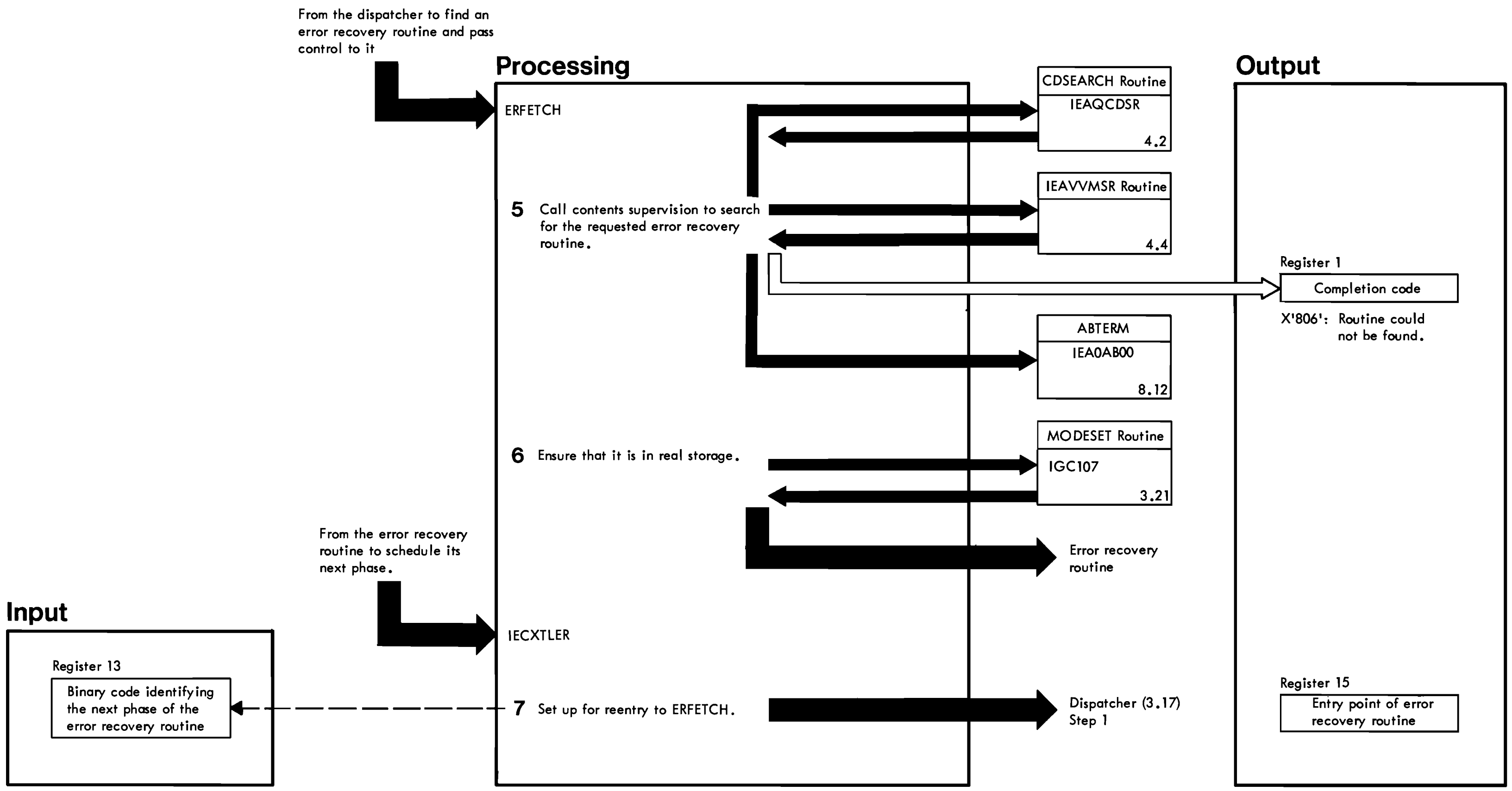
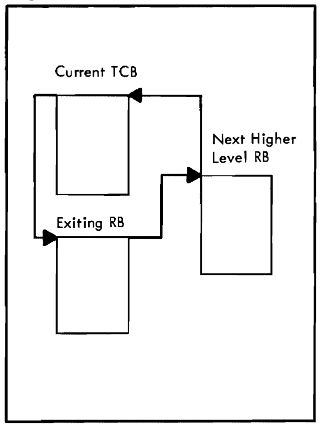

From the dispatcher to find an error recovery routine and pass control to it
Processing
ERFETCH
5 Call contents supervision to search for the requested error recovery routine.
6 Ensure that it is in real storage.
From the error recovery routine to schedule its next phase.
IECXTLER
7 Set up for reentry to ERFETCH.
CDSEARCH Routine
IEAQCDSR
4.2
IEAVVMSR Routine
4.4
ABTERM
IEA0AB00
8.12
MODESET Routine
IGC107
3.21
Error recovery routine
Dispatcher (3.17) Step 1
Input
Register 13
Binary code identifying the next phase of the error recovery routine
Output
Register 1
Completion code
X'806': Routine could not be found.
Register 15
Entry point of error recovery routine

Diagram 3.13 (Steps 5-7) Stage 3 Exit Effector (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
7 Successive phases of error routines are scheduled by the Stage 3 Exit Effector; a system error routine branches to IECXTLER, which causes ERFETCH to be reentered to search for the next phase.	Stage 3 Exit Effector	IECXTLER

From user or system programs (except type-1 SVCs) to handle exiting from these programs

Input



Processing

IGC003

1 If a caller is a user program check routine, see the notes for this step.

2 Free STA control blocks.

3 Perform special processing based on the type of RB that represents the completed program.

Last program for the task is exiting → End-of-Task (EOT) Routine 8.3

4 Adjust the dispatchability of the task as required.

5 If the next higher level RB is waiting, or the task is not dispatchable, force a task switch.

6 Free the RB of the completed program. → Dispatcher (3.17) Step 1

Output

IEATCBP=0

Diagram 3.14 Exit Routine (Module IEAVET00)

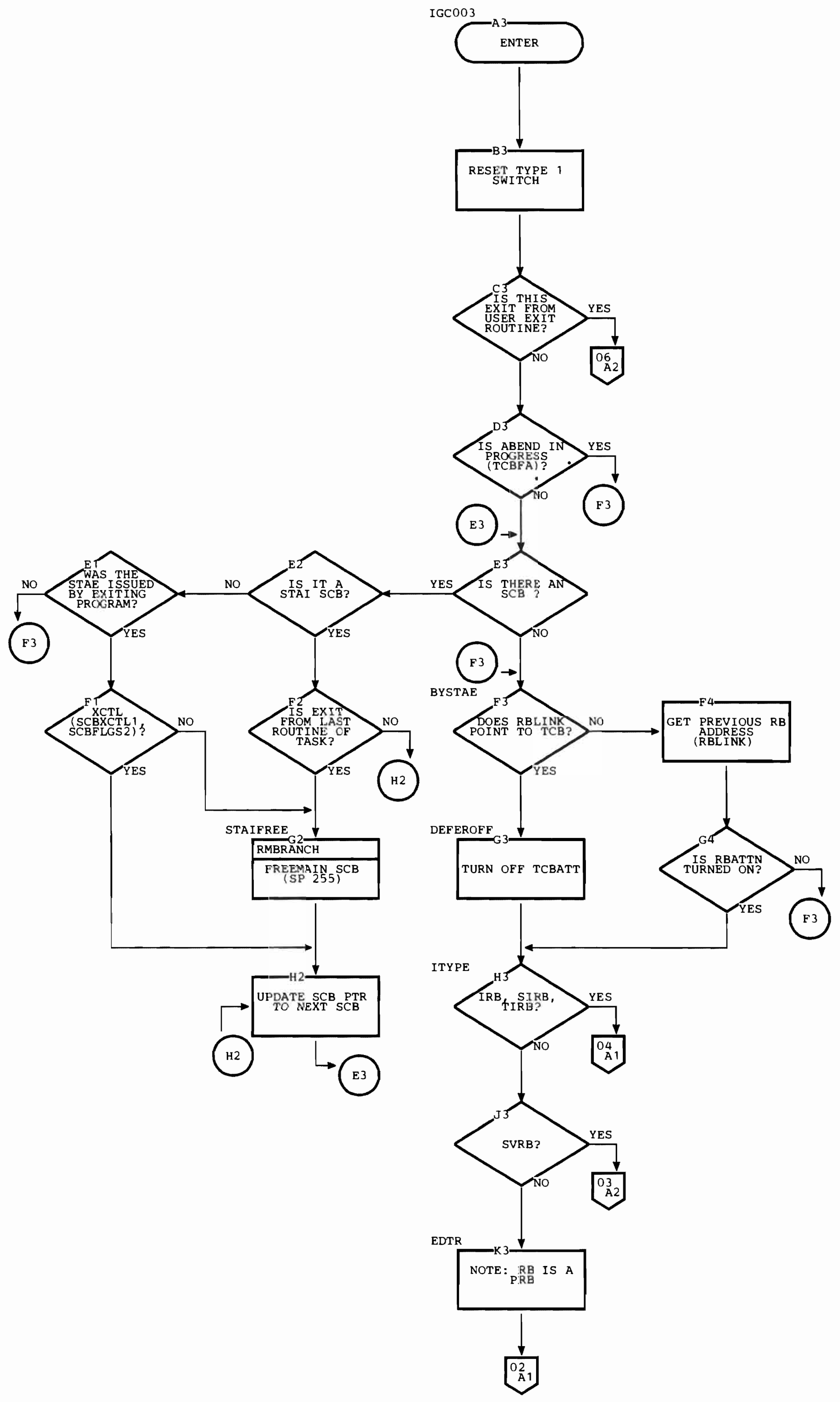
NOTES	ROUTINE NAME	LABEL
The Exit routine, a type-1 SVC routine, handles the exiting procedures for programs other than type-1 SVC routines. Problem programs or system programs gain supervisor-assisted linkage to the Exit routine by issuing a RETURN macro instruction; type-1 SVC routines obtain a similar result by using an SVC 3 instruction. The Exit routine determines the type of program that is exiting. 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 TIRB (task interruption request block). For each type of exiting program, some special processing is performed. If the completed program was the first-executed program of its task, and therefore is considered to be at the "highest control level" within that task, the Exit routine recognizes an end-of-task condition and branches to the End-of-Task routine (EOT) to perform normal termination of the caller's 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.	Exit Routine	IGC003
1 User program-check routine (no RB) • Ensure that PIE is in real storage. • Restore interrupted routine's registers from low storage 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 interruption FLIH. An active PIE leads FLIH 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.		EDUX

NOTES	ROUTINE NAME	LABEL
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. The Exit routine branches to the dispatcher, which returns control to the interrupted user program.	Exit	
2 If the returning program is not a user program-check routine, the Exit routine examines the SCB (STA control block) queue pointed to by the TCBNSTAE field in the TCB. If there are no SCBs or if ABEND is in progress, the SCB processing is bypassed and Exit determines the RB type. Exit removes from the queue and frees all SCBs for the exiting program except those SCBs for which the exiting program issued a STAE macro instruction with the XCTL option. The search of the queue terminates when an SCB is found for an RB other than the RB for the exiting program. When the RB for the program that is exiting is the last PRB, Exit frees all SCBs, including those for STAI.		
3 Exit determines the type of program that is exiting 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.		
User program (PRB) • Restore user's registers from low storage to the TCB general register save area. • If the returning user program is the last to be executed for the task, branch to the EOT (End-of-Task) routine to perform normal task termination. Exit routine branches to EOT routine. • Go to CDEXIT to determine whether there are other requests for use of the completed program, and to prepare for re-entry to the program if there are such requests. After CDEXIT, execution continues at Step 4.		EDTR
• CDEXIT tests whether the exiting program has a CDE (contents directory entry); the existence of a CDE is indicated in the CDE field of the PRB. If there is no CDE, the exiting program was entered via the SYNCH macro instruction, which does not build a CDE; in this case, the CDEXIT routine returns control to Exit. If there is a CDE, CDEXIT continues processing.		CDEXIT

NOTES	ROUTINE NAME	LABEL
CDEXIT finds the major CDE associated with the PRB of the complete routine, and then reduces the use/responsibility count in the major CDE. CDEXIT updates the RB address in the CDE so it points to the next PRB that will control the program. This next PRB is associated with a task different from that of the caller.	Exit	
CDEXIT makes the new PRB ready by placing 0 in its wait count field; the dispatcher will test this field before dispatching the program. CDEXIT also sets the right half of the old PSW field in the new PRB, in preparation for later entry to the contents supervision at IEAQCS03 in CDEPILOG (Diagram 4.2).		
The CDEPILOG subroutine is executed when the dispatcher recognizes the new PRB's task as the highest priority ready task. (CDEPILOG performs final preparation for linkage to the requested program.)		
After preparing the next PRB to control the program, CDEXIT branches to the Task Switch routine. This routine tests whether the TCB for the previously waiting PRB may replace the current TCB. The Task Switch routine returns control to CDEXIT.		
If there are no other requests for the exiting program, CDEXIT uses its subroutine, the CDHKEEP routine. CDHKEEP sets the "nonfunctional" flag in the CDE to indicate that the program has been executed. Although this flag is meaningful only for nonreusable programs, it is always set at this point.		CDHKEEP
CDHKEEP tests the use/responsibility count in the CDE to determine whether there are other requests for the exiting program. If the use/responsibility count is not 0, there is at least one outstanding request for the program, and CDHKEEP returns control to Exit (or to CDHKEEP's caller).		
If, however, the use/responsibility count is 0, there is no outstanding request for the program. In this case, the routine tests the program's attributes.		
If the program's storage can be freed, either the CDPURGE subroutine in GETMAIN, or another subroutine of Exit routine (CDDESTRY), frees the storage.		CDDESTRY
Control returns to the Exit routine (Step 4), or another caller.		

NOTES		ROUTINE NAME	LABEL
	Type 2, 3, or 4 SVC routine (SVRB) • Exit restores the registers belonging to the caller of the completed SVC routine from the SVRB register save area to the TCB general register save area. Execution continues at Step 4.	Exit	EDSVRB
	IRB, SIRB, or TIRB • Processing of a completed program controlled by an SIRB continues at Step 4.		SKIPATTN
	IRB or TIRB • If there are additional requests for use of a program represented by an IRB or a TIRB, prepare the program for reuse by setting up the RB old PSW and the TCB general register save area. Exit branches to the dispatcher.		SKIPATTN
	• If there are no additional requests for use of a completed module, Exit calls the WAIT routine at JSTIME, to test if all the RBs in the region are in a wait state without any time limitations. WAIT applies time limits if there are none. Execution continues at Step 4.		WAITEST
	TSO Task with Attention IRB • Start all subtasks via a branch entry to IGC07902 (Diagram 3.18).		IKJATTN
	• Free the TAIE. Execution continues at SKIPATIN, see above.		
4	If the TCBSTPPR flag is set, indicating that a task is to be set nondispatchable when no longer executing a privileged program (RBATTN=0 for all RBs), a search of the RB queue is necessary.		EDNEXT
5	No task switch is made if NEW does not equal OLD already.		
6	Exit then sets the RB for the exiting program inactive and removes it from the RB queue and frees, if possible, the RB area. If the exiting RB is an IRB with a problem program save area, the save area is also freed. Exit then branches directly to the dispatcher.		EDACT

Chart 3-2 (page 1 of 11). Exit routine



IGC003
A3 ENTER
B3 RESET TYPE 1 SWITCH
C3 IS THIS EXIT FROM USER EXIT ROUTINE?
YES
NO
06 A2
D3 IS ABEND IN PROGRESS (TCBFA)?
YES
NO
F3
E3
E3 IS THERE AN SCB ?
YES
NO
E2 IS IT A STAI SCB?
NO
YES
E1 WAS THE STAE ISSUED BY EXITING PROGRAM?
NO
YES
F3
F1 XCTL (SCBXCTL1, SCBFLGS2)?
NO
YES
F2 IS EXIT FROM LAST ROUTINE OF TASK?
NO
YES
H2
F3
BYSTAE
F3 DOES RBLINK POINT TO TCB?
NO
YES
F4 GET PREVIOUS RB ADDRESS (RBLINK)
G4 IS RBATTN TURNED ON?
NO
YES
F3
STAIFREE
G2 RMBRANCH FREEMAIN SCB (SP 255)
DEFEROFF
G3 TURN OFF TCBATT
ITYPE
H3 IRB, SIRB, TIRB?
YES
NO
04 A1
H2 UPDATE SCB PTR TO NEXT SCB
H2
E3
J3 SVRB?
YES
NO
03 A2
EDTR
K3 NOTE: RB IS A PRB
02 A1

Chart 3-2 (page 2 of 11). Exit routine

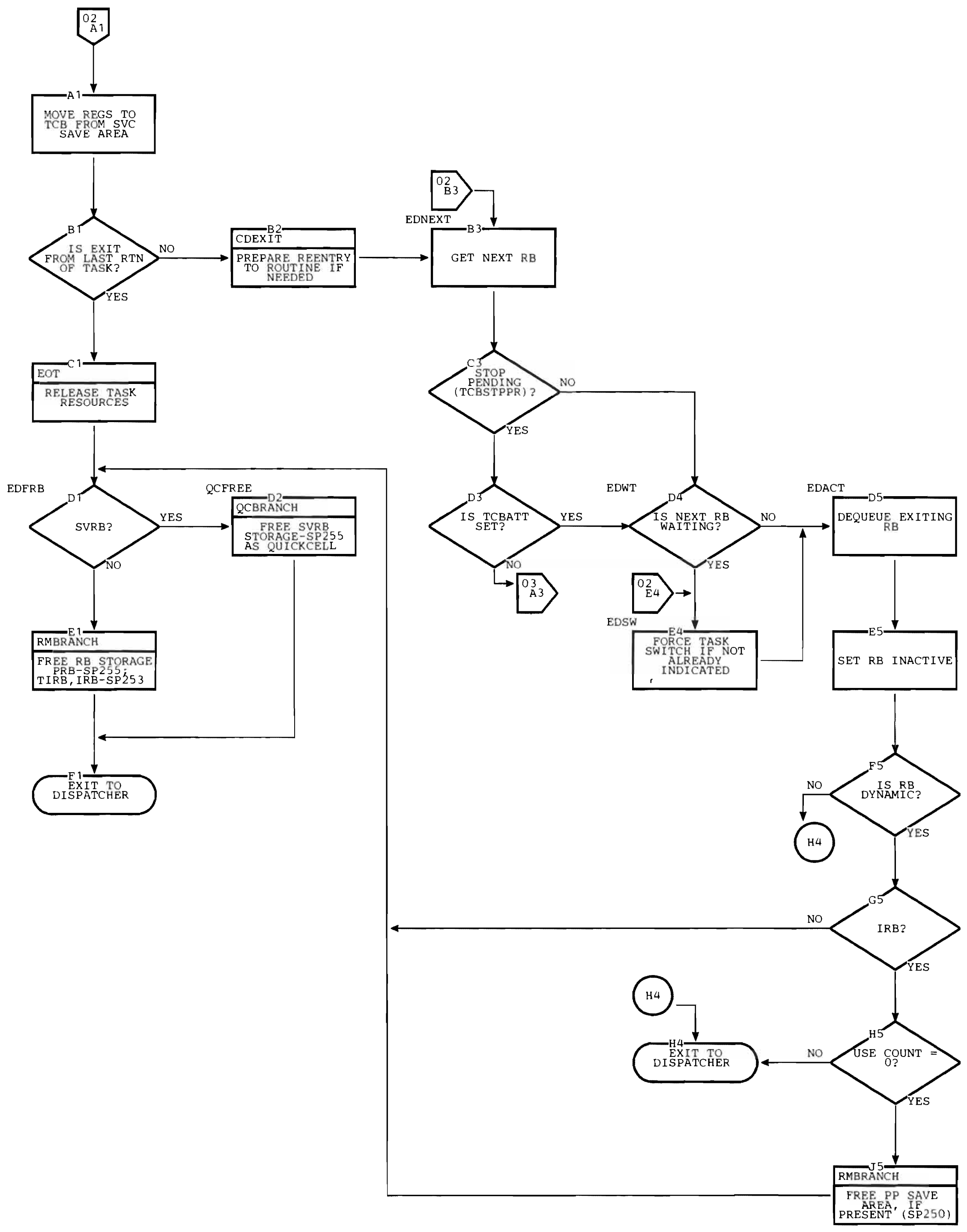


Chart 3-2 (page 3 of 11). Exit routine

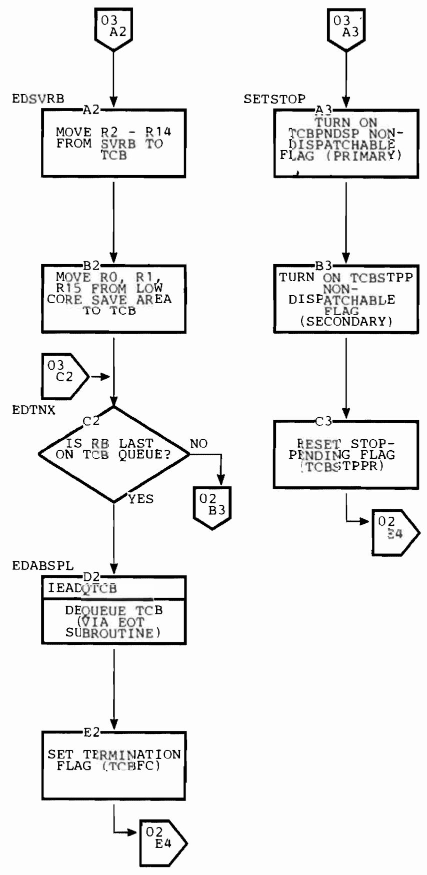

03 A2
EDSVRB
A2
MOVE R2 - R14 FROM SVRB TO TCB
B2
MOVE R0, R1, R15 FROM LOW CORE SAVE AREA TO TCB
03 C2
EDTNX
C2
IS RB LAST ON TCB QUEUE?
NO
02 B3
YES
EDABSPL
D2
IEADQTCB
DEQUEUE TCB (VIA EOT SUBROUTINE)
E2
SET TERMINATION FLAG (TCBFC)
02 E4
03 A3
SETSTOP
A3
TURN ON TCBPNDSP NON-DISPATCHABLE FLAG (PRIMARY)
B3
TURN ON TCBSTPP NON-DISPATCHABLE FLAG (SECONDARY)
C3
RESET STOP-PENDING FLAG (TCBSTPPR)
02 E4

Chart 3-2 (page 4 of 11). Exit routine

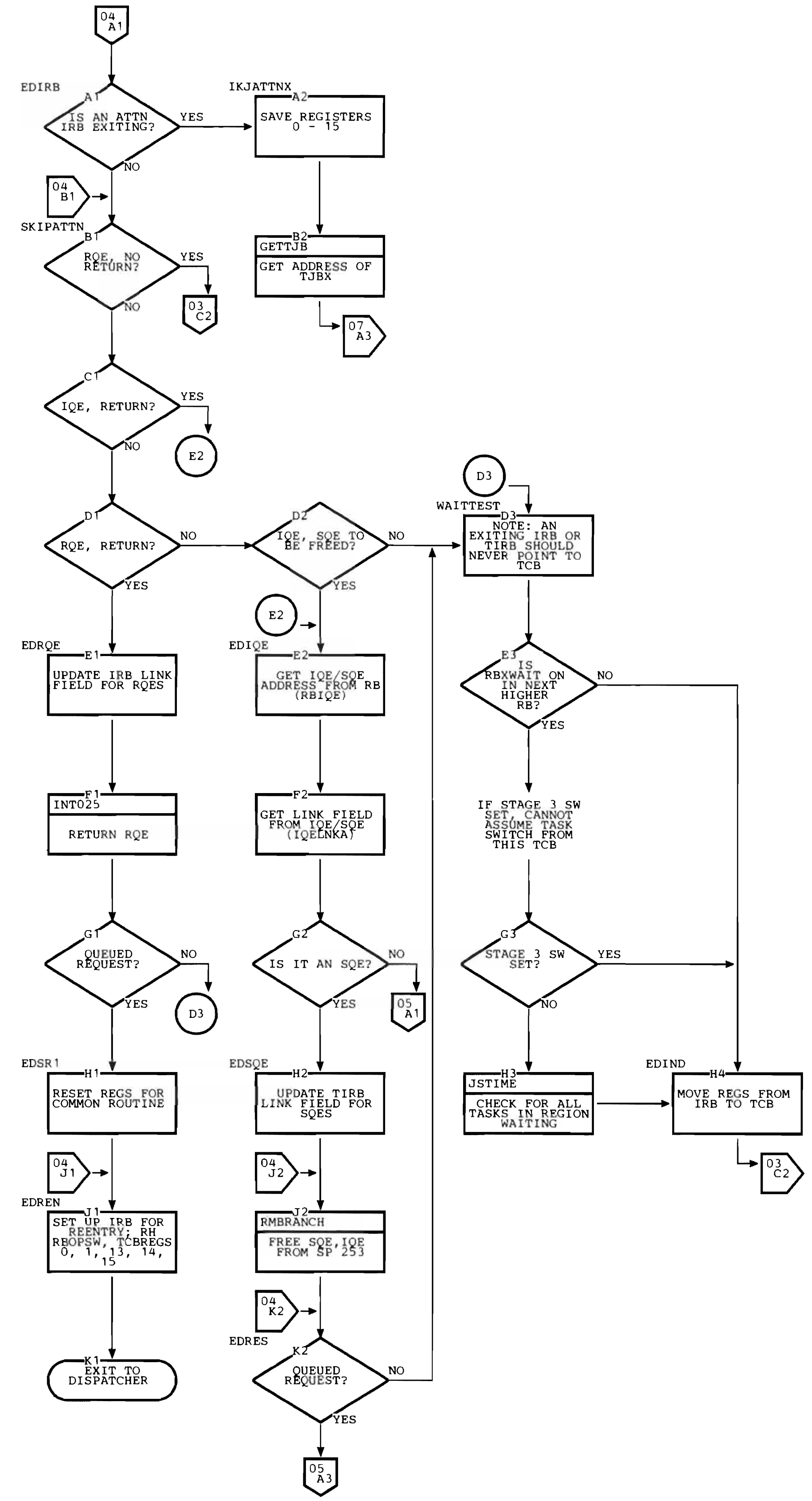
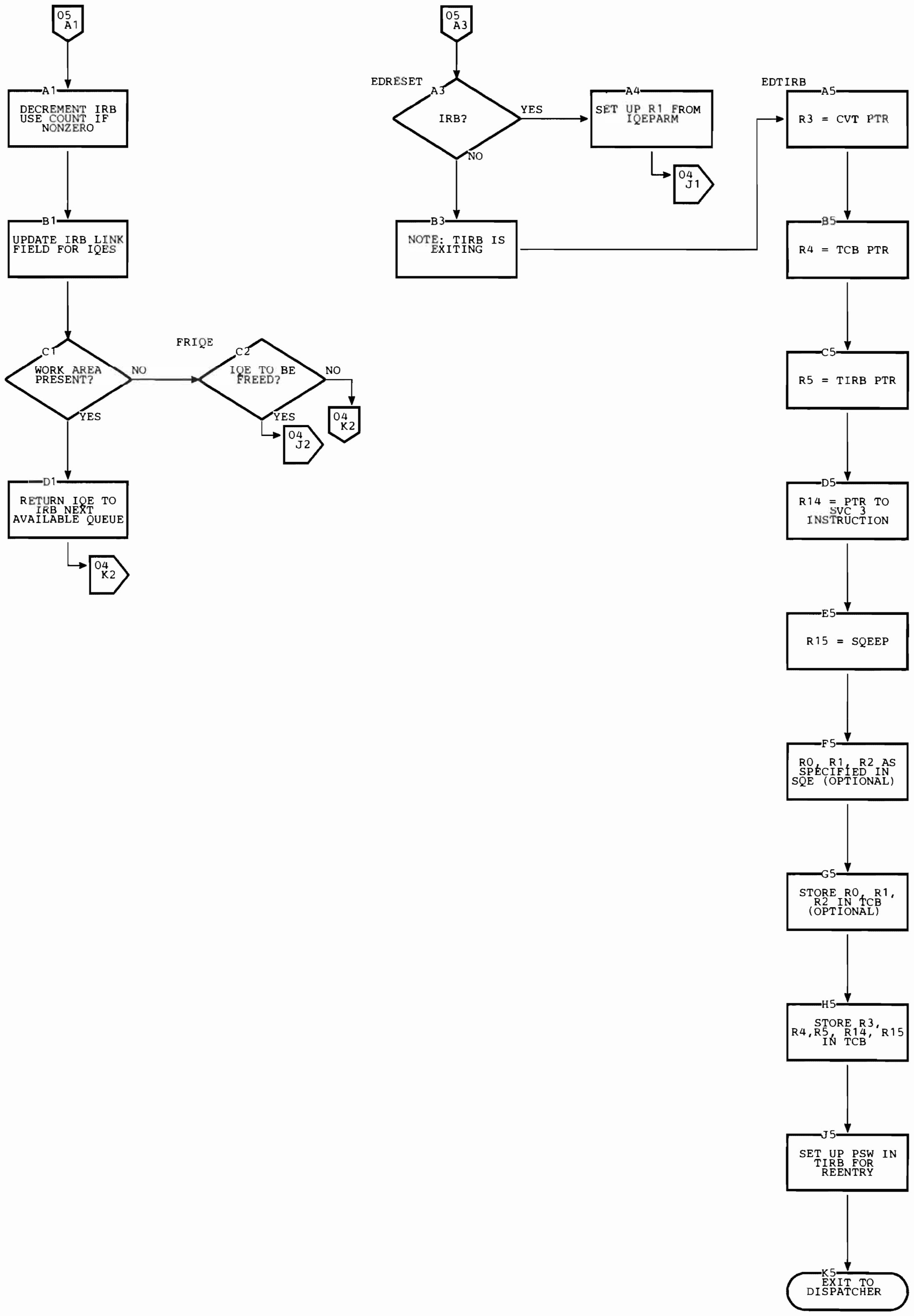
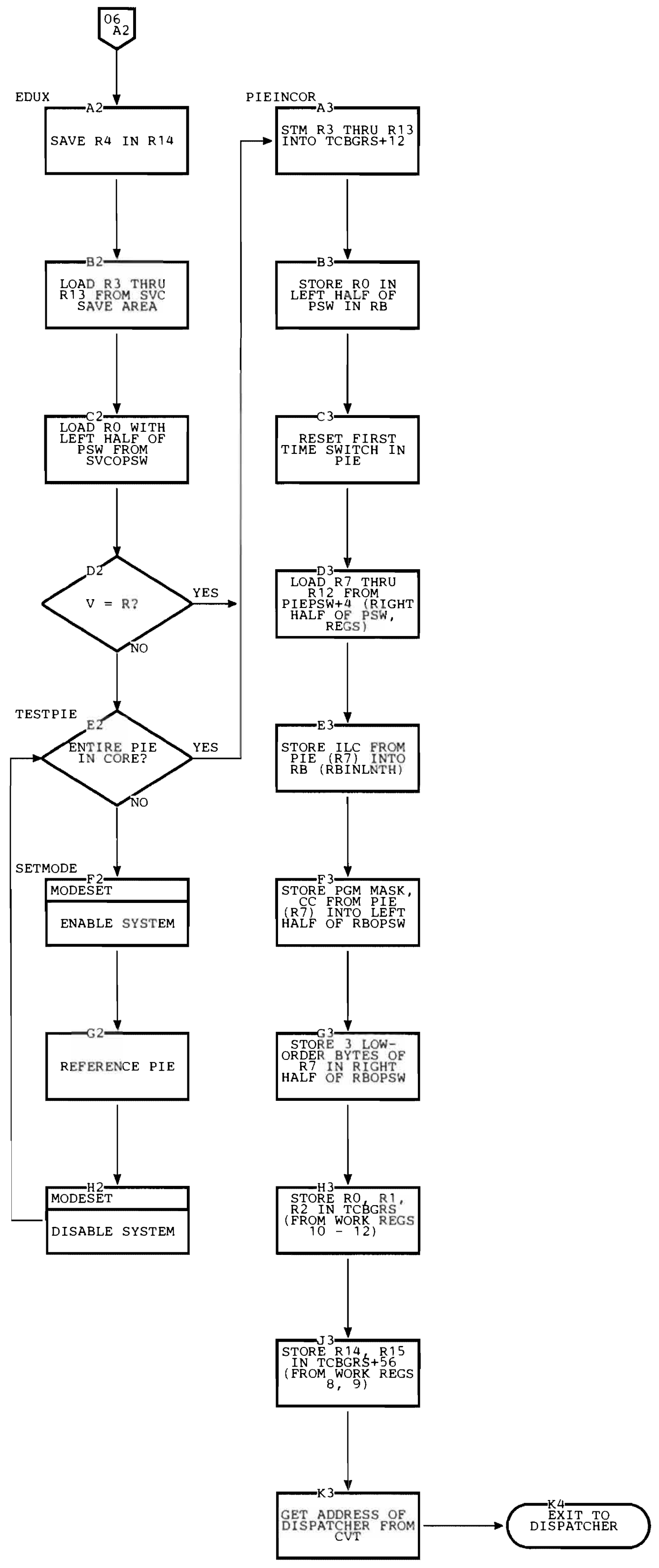

04 A1
EDIRB
A1 IS AN ATTN IRB EXITING?
YES
NO
IKJATTNX
A2 SAVE REGISTERS 0 - 15
04 B1
SKIPATTN
B1 RQE, NO RETURN?
YES
NO
03 C2
B2 GETTJB
GET ADDRESS OF TJBX
07 A3
C1 IQE, RETURN?
YES
NO
E2
D3
WAITTEST
D1 RQE, RETURN?
NO
YES
D2 IQE, SQE TO BE FREED?
NO
YES
D3 NOTE: AN EXITING IRB OR TIRB SHOULD NEVER POINT TO TCB
E2
EDRQE
E1 UPDATE IRB LINK FIELD FOR RQES
EDIQE
E2 GET IQE/SQE ADDRESS FROM RB (RBIQE)
E3 IS RBXWAIT ON IN NEXT HIGHER RB?
NO
YES
F1 INT025
RETURN RQE
F2 GET LINK FIELD FROM IQE/SQE (IQELNKA)
IF STAGE 3 SW SET, CANNOT ASSUME TASK SWITCH FROM THIS TCB
G1 QUEUED REQUEST?
NO
YES
D3
G2 IS IT AN SQE?
NO
YES
05 A1
G3 STAGE 3 SW SET?
YES
NO
EDSR1
H1 RESET REGS FOR COMMON ROUTINE
EDSQE
H2 UPDATE TIRB LINK FIELD FOR SQES
H3 JSTIME
CHECK FOR ALL TASKS IN REGION WAITING
EDIND
H4 MOVE REGS FROM IRB TO TCB
04 J1
04 J2
03 C2
EDREN
J1 SET UP IRB FOR REENTRY; RH RBOPSW, TCBREGS 0, 1, 13, 14, 15
J2 RMBRANCH
FREE SQE,IQE FROM SP 253
04 K2
EDRES
K1 EXIT TO DISPATCHER
K2 QUEUED REQUEST?
NO
YES
05 A3

Chart 3-2 (page 5 of 11). Exit routine

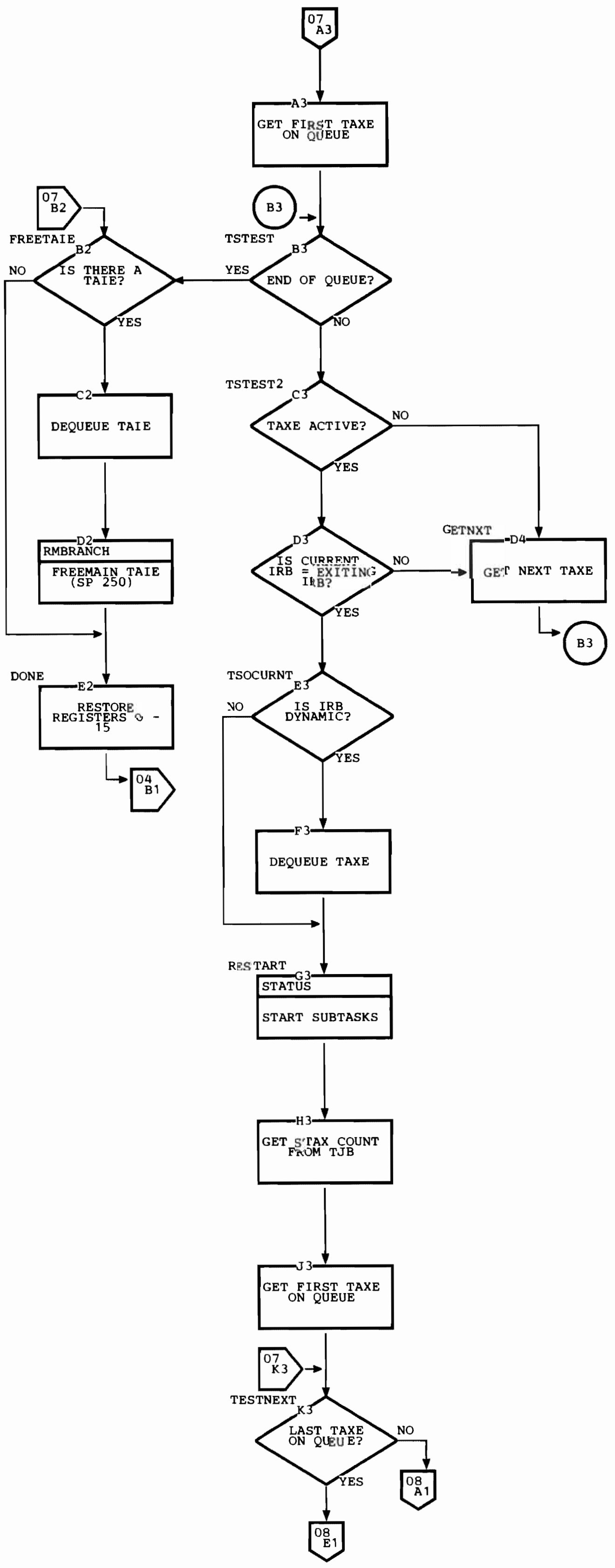


05 A1
A1 DECREMENT IRB USE COUNT IF NONZERO
B1 UPDATE IRB LINK FIELD FOR IQES
C1 WORK AREA PRESENT?
NO
YES
FRIQE
C2 IQE TO BE FREED?
NO
YES
04 J2
04 K2
D1 RETURN IQE TO IRB NEXT AVAILABLE QUEUE
04 K2
05 A3
EDRESET
A3 IRB?
YES
NO
A4 SET UP R1 FROM IQEPARM
04 J1
B3 NOTE: TIRB IS EXITING
EDTIRB
A5 R3 = CVT PTR
B5 R4 = TCB PTR
C5 R5 = TIRB PTR
D5 R14 = PTR TO SVC 3 INSTRUCTION
E5 R15 = SQEEP
F5 R0, R1, R2 AS SPECIFIED IN SQE (OPTIONAL)
G5 STORE R0, R1, R2 IN TCB (OPTIONAL)
H5 STORE R3, R4,R5, R14, R15 IN TCB
J5 SET UP PSW IN TIRB FOR REENTRY
K5 EXIT TO DISPATCHER

Chart 3-2 (page 6 of 11). Exit routine


06 A2
EDUX A2
SAVE R4 IN R14
B2
LOAD R3 THRU R13 FROM SVC SAVE AREA
C2
LOAD R0 WITH LEFT HALF OF PSW FROM SVCOPSW
D2
V = R?
YES
NO
TESTPIE E2
ENTIRE PIE IN CORE?
YES
NO
SETMODE F2
MODESET
ENABLE SYSTEM
G2
REFERENCE PIE
H2
MODESET
DISABLE SYSTEM
PIEINCOR A3
STM R3 THRU R13 INTO TCBGRS+12
B3
STORE R0 IN LEFT HALF OF PSW IN RB
C3
RESET FIRST TIME SWITCH IN PIE
D3
LOAD R7 THRU R12 FROM PIEPSW+4 (RIGHT HALF OF PSW, REGS)
E3
STORE ILC FROM PIE (R7) INTO RB (RBINLNTH)
F3
STORE PGM MASK, CC FROM PIE (R7) INTO LEFT HALF OF RBOPSW
G3
STORE 3 LOW-ORDER BYTES OF R7 IN RIGHT HALF OF RBOPSW
H3
STORE R0, R1, R2 IN TCBGRS (FROM WORK REGS 10 - 12)
J3
STORE R14, R15 IN TCBGRS+56 (FROM WORK REGS 8, 9)
K3
GET ADDRESS OF DISPATCHER FROM CVT
K4
EXIT TO DISPATCHER

• Chart 3-2 (page 7 of 11). Exit routine



• **Chart 3-2 (page 8 of 11). Exit routine**

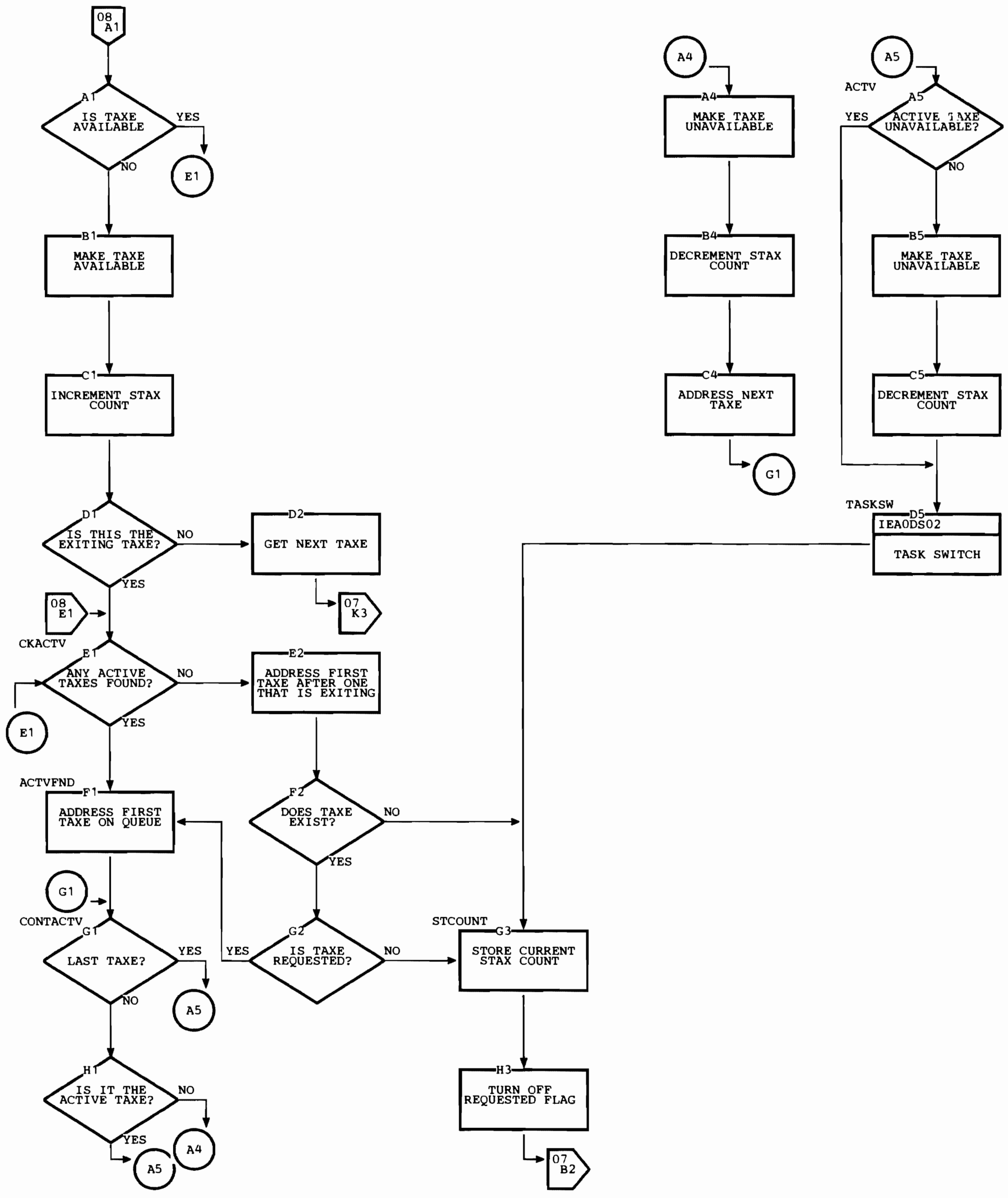


Chart 3-2 (page 9 of 11). Exit routine

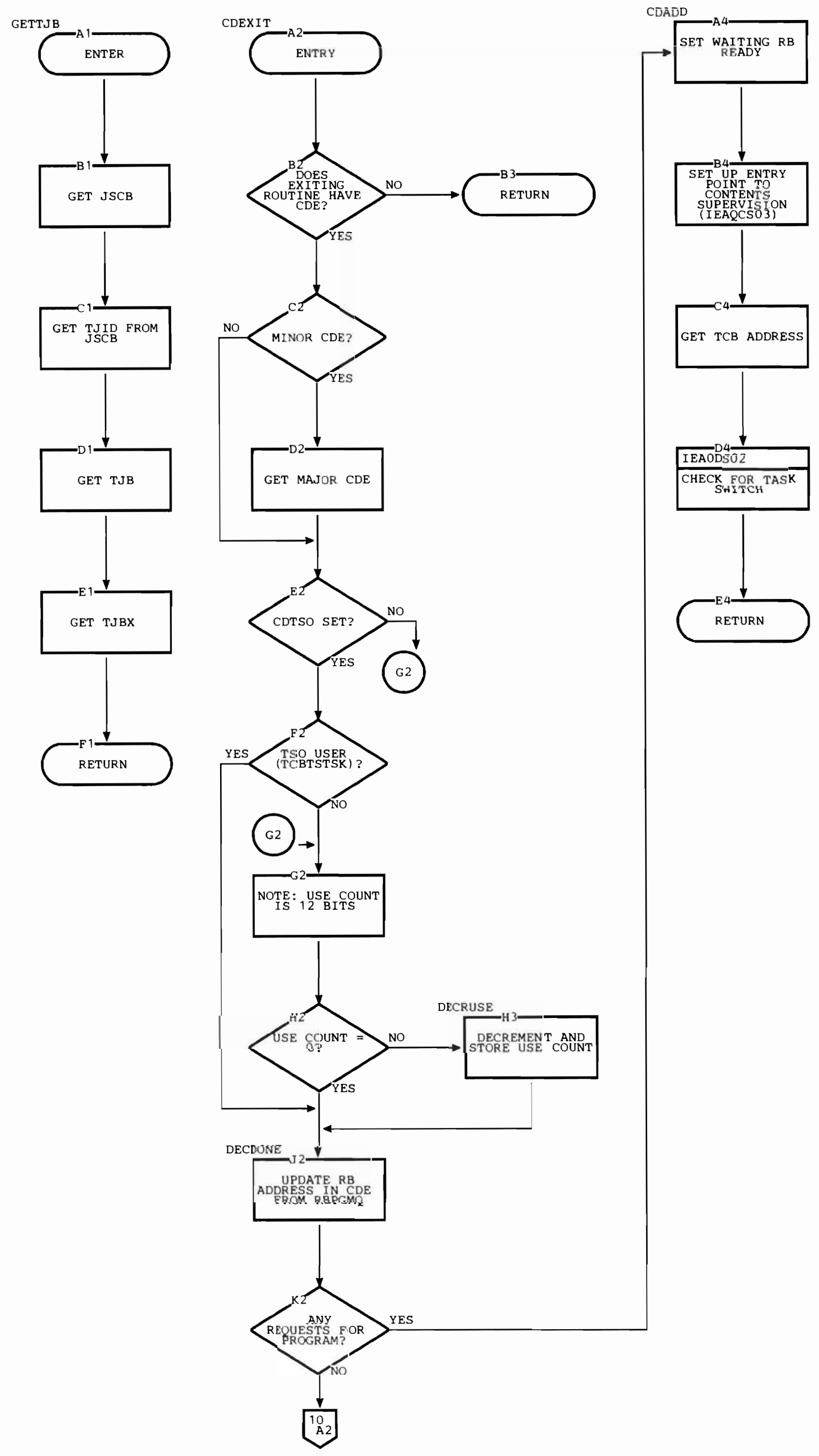
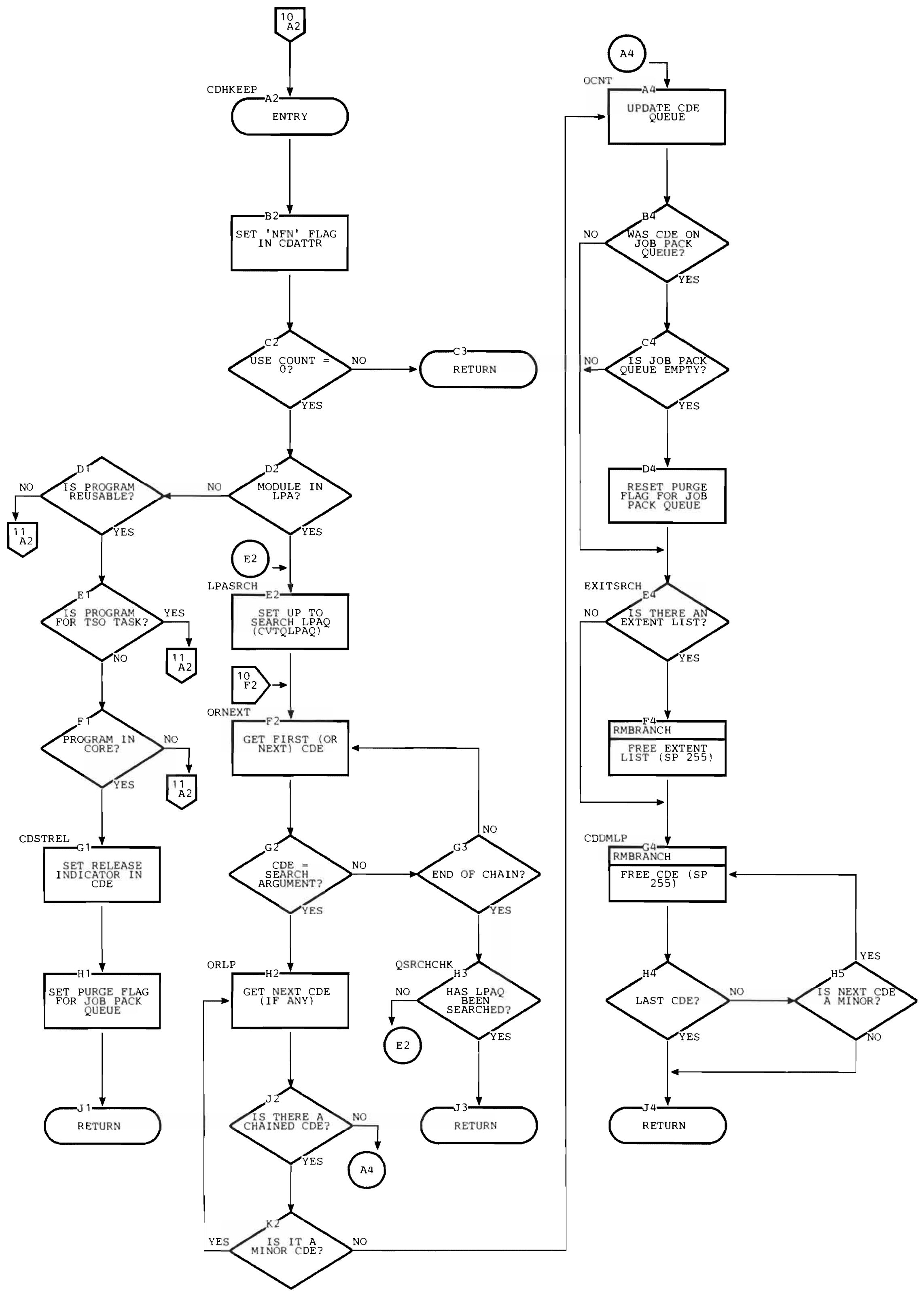


Chart 3-2 (page 10 of 11). Exit routine



10 A2
CDHKEEP A2 ENTRY
B2 SET 'NFN' FLAG IN CDATTR
C2 USE COUNT = 0?
NO
C3 RETURN
YES
D2 MODULE IN LPA?
NO
D1 IS PROGRAM REUSABLE?
NO
11 A2
YES
E1 IS PROGRAM FOR TSO TASK?
YES
11 A2
NO
F1 PROGRAM IN CORE?
NO
11 A2
YES
CDSTREL G1 SET RELEASE INDICATOR IN CDE
H1 SET PURGE FLAG FOR JOB PACK QUEUE
J1 RETURN
YES
E2
LPASRCH E2 SET UP TO SEARCH LPAQ (CVTQLPAQ)
10 F2
ORNEXT F2 GET FIRST (OR NEXT) CDE
G2 CDE = SEARCH ARGUMENT?
NO
G3 END OF CHAIN?
NO
YES
QSRCHCHK H3 HAS LPAQ BEEN SEARCHED?
NO
E2
YES
J3 RETURN
YES
ORLP H2 GET NEXT CDE (IF ANY)
J2 IS THERE A CHAINED CDE?
NO
A4
YES
K2 IS IT A MINOR CDE?
YES
NO
A4
OCNT A4 UPDATE CDE QUEUE
B4 WAS CDE ON JOB PACK QUEUE?
NO
YES
C4 IS JOB PACK QUEUE EMPTY?
NO
YES
D4 RESET PURGE FLAG FOR JOB PACK QUEUE
EXITSRCH E4 IS THERE AN EXTENT LIST?
NO
YES
F4 RMBRANCH FREE EXTENT LIST (SP 255)
CDDMLP G4 RMBRANCH FREE CDE (SP 255)
H4 LAST CDE?
NO
H5 IS NEXT CDE A MINOR?
YES
NO
YES
J4 RETURN

• Chart 3-2 (page 11 of 11). Exit routine

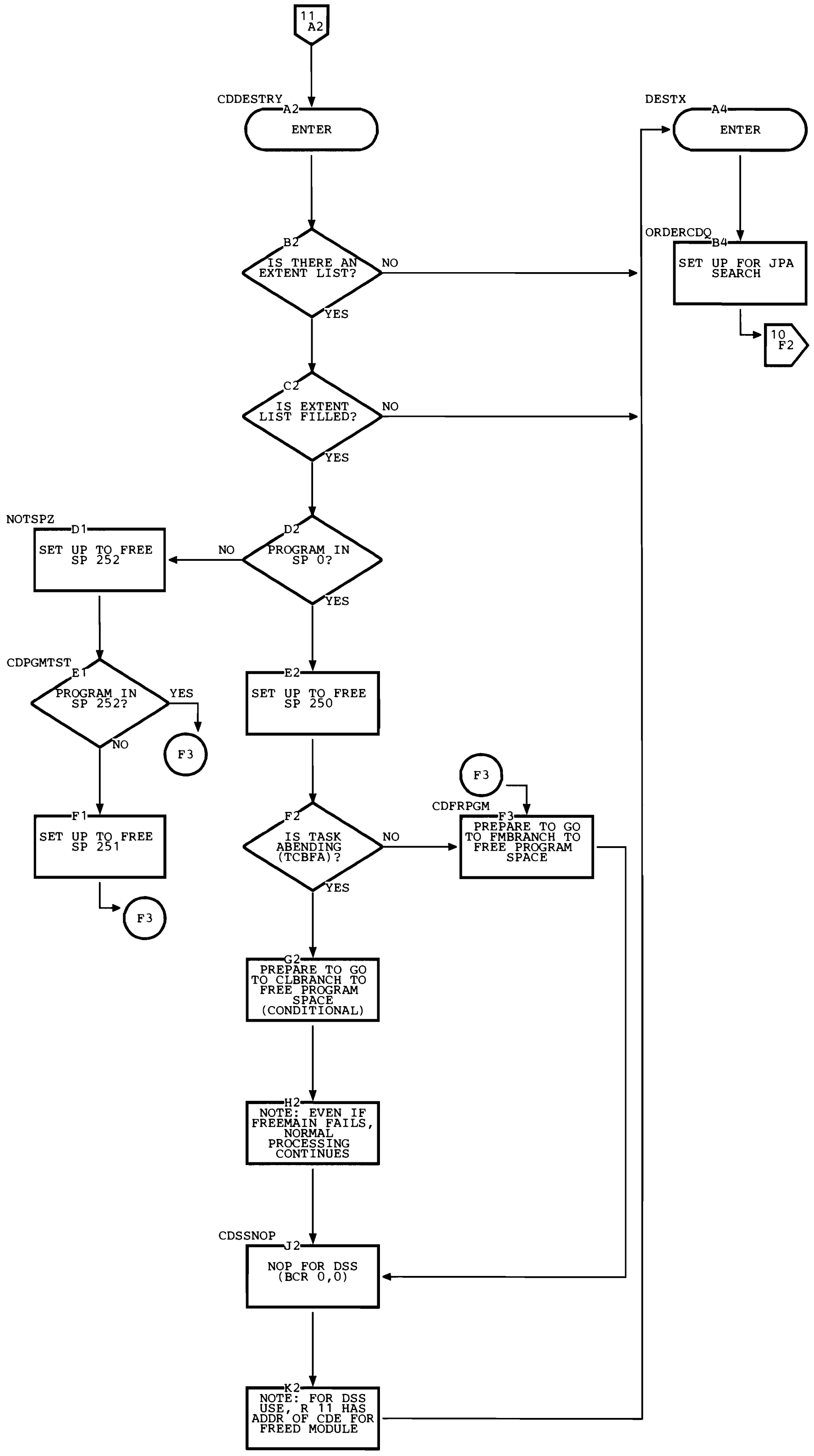
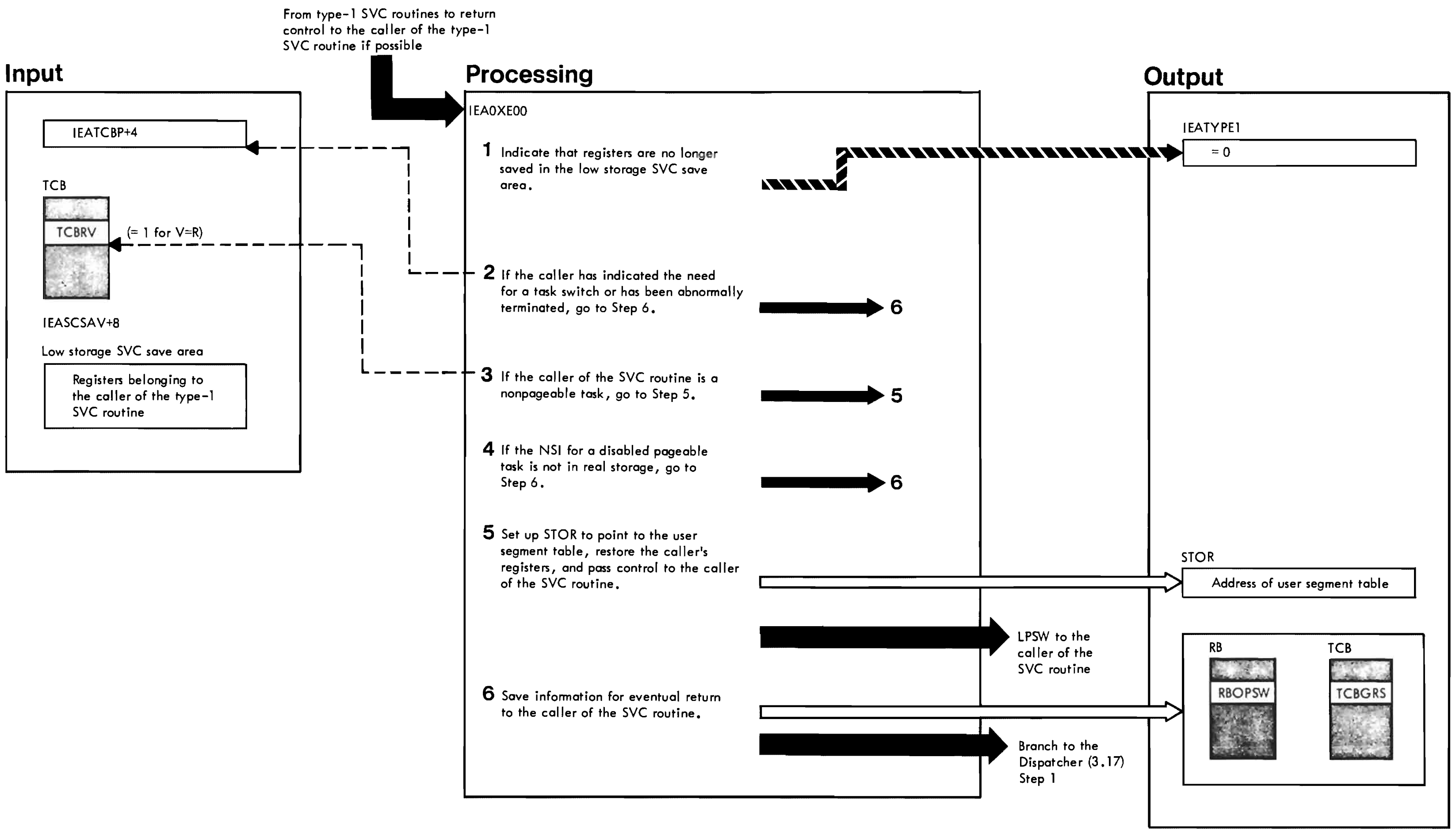

11 A2
CDDESTRY A2 ENTER
B2 IS THERE AN EXTENT LIST?
NO
YES
C2 IS EXTENT LIST FILLED?
NO
YES
NOTSPZ D1 SET UP TO FREE SP 252
D2 PROGRAM IN SP 0?
NO
YES
CDPGMTST E1 PROGRAM IN SP 252?
YES
NO
F3
E2 SET UP TO FREE SP 250
F1 SET UP TO FREE SP 251
F3
F3
CDFRPGM F3 PREPARE TO GO TO FMBRANCH TO FREE PROGRAM SPACE
F2 IS TASK ABENDING (TCBFA)?
NO
YES
G2 PREPARE TO GO TO CLBRANCH TO FREE PROGRAM SPACE (CONDITIONAL)
H2 NOTE: EVEN IF FREEMAIN FAILS, NORMAL PROCESSING CONTINUES
CDSSNOP J2 NOP FOR DSS (BCR 0,0)
K2 NOTE: FOR DSS USE, R 11 HAS ADDR OF CDE FOR FREED MODULE
DESTX A4 ENTER
ORDERCDQ B4 SET UP FOR JPA SEARCH
10 F2

Diagram 3.15 Type-1 Exit Routine


From type-1 SVC routines to return control to the caller of the type-1 SVC routine if possible
Input
IEATCBP+4
TCB
TCBRV
(= 1 for V=R)
IEASCSAV+8
Low storage SVC save area
Registers belonging to the caller of the type-1 SVC routine
Processing
IEA0XE00
6
5
6
LPSW to the caller of the SVC routine
Branch to the Dispatcher (3.17) Step 1
Output
IEATYPE1
= 0
STOR
Address of user segment table
RB
RBOPSW
TCB
TCBGRS

1 Indicate that registers are no longer saved in the low storage SVC save area.

2 If the caller has indicated the need for a task switch or has been abnormally terminated, go to Step 6.

3 If the caller of the SVC routine is a nonpageable task, go to Step 5.

4 If the NSI for a disabled pageable task is not in real storage, go to Step 6.

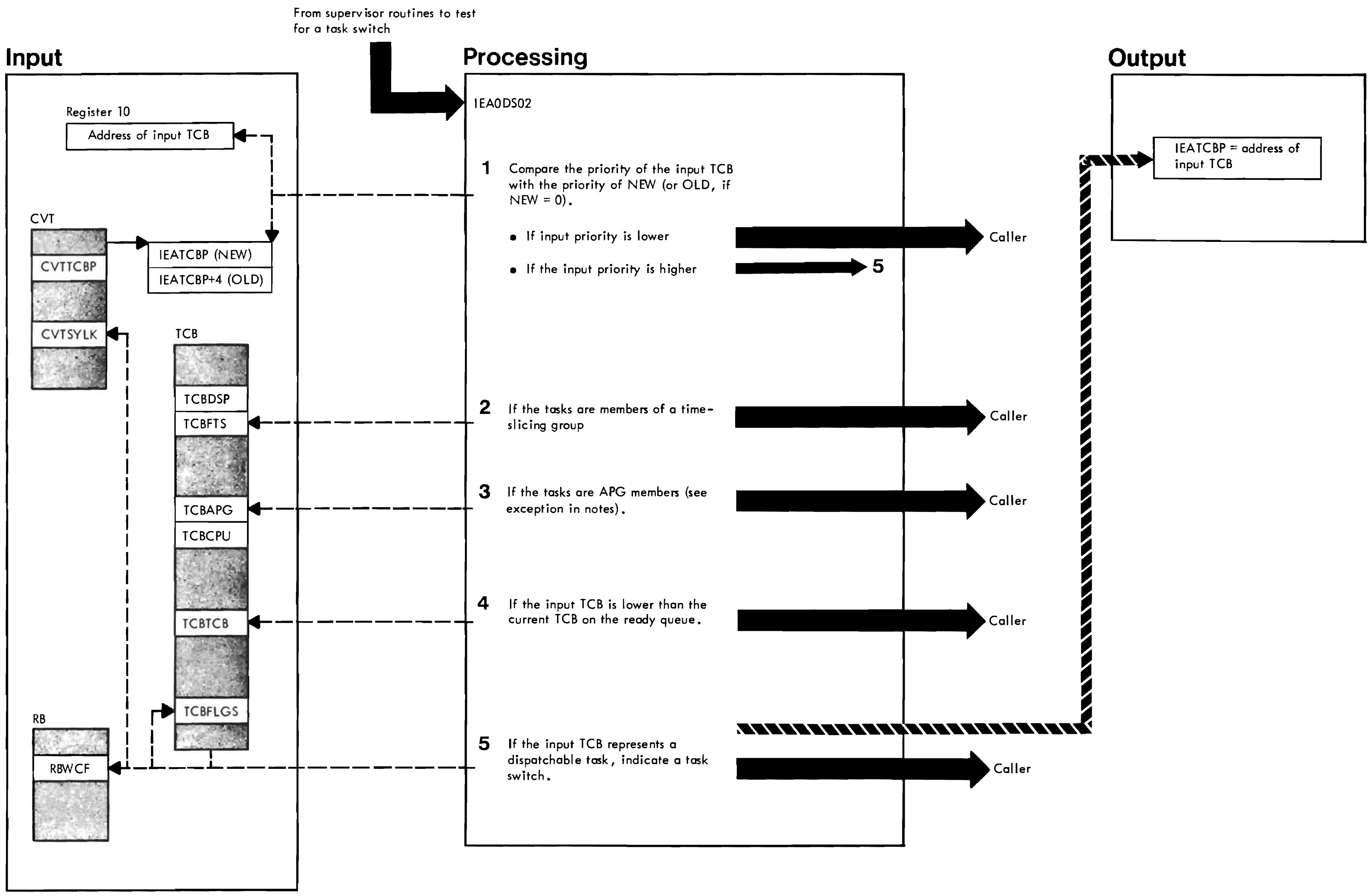
5 Set up STOR to point to the user segment table, restore the caller's registers, and pass control to the caller of the SVC routine.

6 Save information for eventual return to the caller of the SVC routine.

Diagram 3.15 Type-1 Exit Routine (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
The Type-1 Exit routine returns control following execution of a type-1 SVC routine. It determines whether control should be returned directly to the caller of the SVC routine or to the dispatcher. It passes control to the dispatcher if the SVC routine has indicated the need for a task switch by altering the "new" TCB pointer, IEATCBP, or if a disabled page fault would be incurred by passing control directly to the caller.	Type-1 Exit	IEA0XE00
1 Type-1 Exit is entered from any type-1 SVC routine via a branch. Its first step is to set the type-1 switch to zero. ABTERM uses this switch to determine whether a type-1 SVC routine called it.		
2 Some type-1 SVC routines can place a task in a wait condition or make a program ready, thus requiring a task switch. The Type-1 Exit routine tests IEATCBP and IEATCB+4. If the pointers are equal, no task switch is required.		
If the task is being abnormally terminated, control also goes to the dispatcher.		
4 This branch to the dispatcher should not occur often, because it means a disabled routine was exposed to page faults. The dispatcher handles this special condition.		
If the caller of the SVC routine is a disabled task and the NSI is in real storage, or it is a pageable task but is not disabled, the Type-1 Exit routine returns control to it.		

Diagram 3.16
Task Switch Routine



1 Compare the priority of the input TCB with the priority of NEW (or OLD, if NEW = 0).

- If input priority is lower
- If the input priority is higher

2 If the tasks are members of a time-slicing group

3 If the tasks are APG members (see exception in notes).

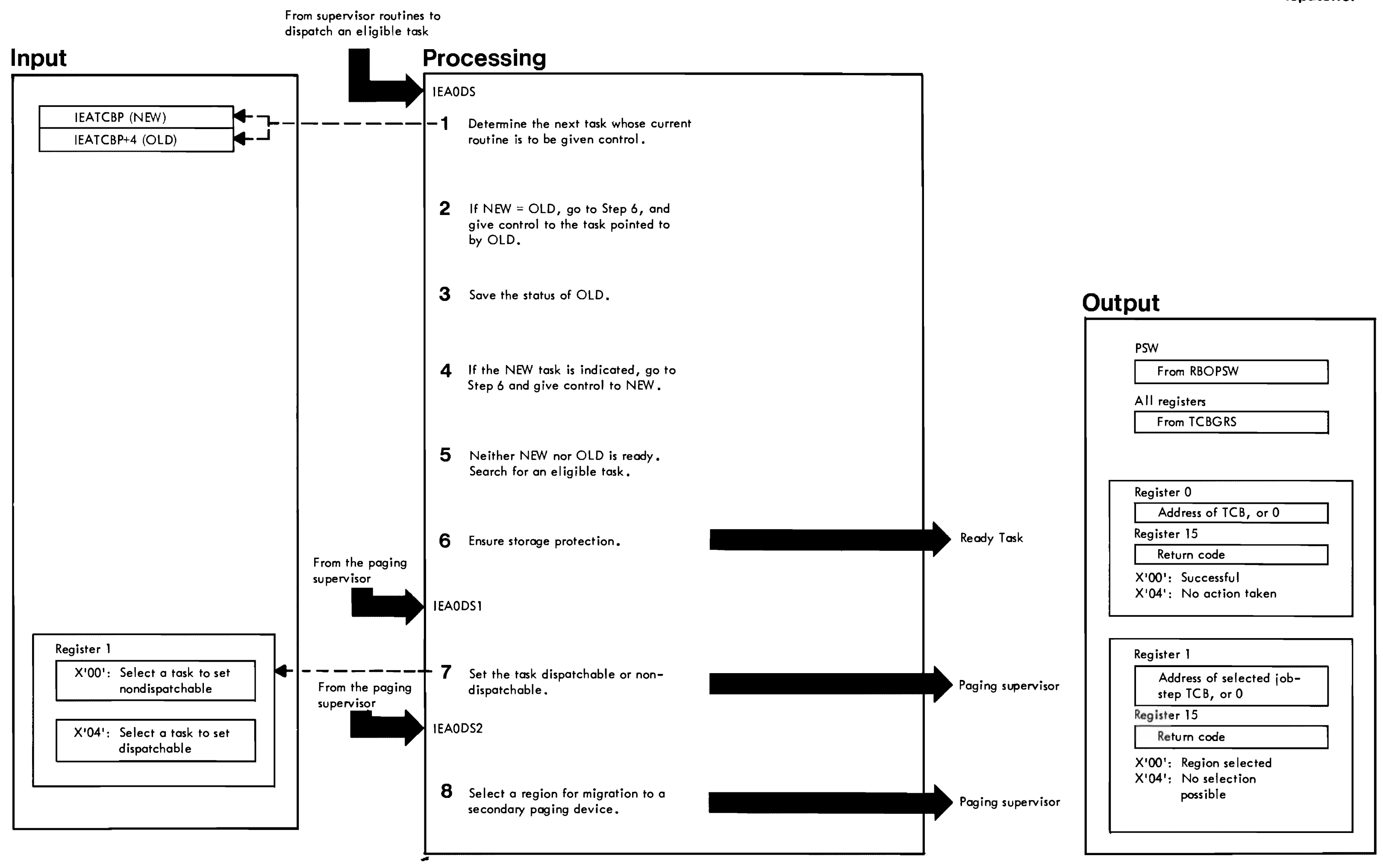
4 If the input TCB is lower than the current TCB on the ready queue.

5 If the input TCB represents a dispatchable task, indicate a task switch.

•Diagram 3.16 The Task Switch Routine (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
The Task Switch routine is used by any disabled supervisor routine that wants to determine whether a newly readied task, which may be of higher priority than the caller of the supervisor routine, should be dispatched in place of the caller's task. A supervisor routine can enter the Task Switch routine after it has reduced a program's RB wait count to 0, or after it has cleared a nondispatchability flag in a TCB. For example, the POST routine may make ready a program that was awaiting the completion of an I/O operation, or the DEQ routine may make ready a program that was awaiting a serially reusable resource. In either case, the supervisor routine does not know if the readied routine belongs to a task of higher priority than that of the caller, and does not know whether it should replace the caller as the currently dispatchable program. The supervisor routines use the Task Switch routine to resolve these questions.	Task Switch	IEA0DS02
1 First, Task Switch checks IEATCBP, the NEW TCB pointer. Normally IEATCBP contains the address of the TCB belonging to the task that will be next dispatched. IEATCBP points either to the TCB of the caller's task, to the TCB of a previously readied task, or is 0. If IEATCBP is not 0, Task Switch compares the priority in the TCB that IEATCB points to with the priority in the input TCB. If the first word of the TCB pointer equals 0, Task Switch compares the dispatching priority of the newly readied routine to the priority in the TCB pointed to by IEATCBP+4, the OLD TCB pointer.		
2 The Task Switch routine tests the time-slicing bit in the TCBs. If either task is a member of a time-slicing group (TCBFTS = 1), Task Switch cannot indicate a task switch.		
3 If Task Switch is comparing two tasks that are members of an APG (automatic priority group; TCBAPG = 1), and if the newly readied task is I/O oriented (TCBCPU = 0) and the comparison task is CPU oriented (TCBCPU = 1), Task Switch may signal a task switch (based on Step 5).	Task Switch	
4 Task Switch examines the task queue, beginning with the comparison task's TCB. If it finds a task of lower priority than the input task before it reaches the input task's TCB, or reaches the end of the task queue before it finds the input task's TCB, Task Switch may signal a task switch (based on Step 5).		SWNEXT
5 The following tests determine whether the TCB address of the input task can be placed in IEATCBP, and thus brought to the attention of the dispatcher: • Task must be dispatchable (TCBFLGS=0). • Highest RB level of the task cannot be waiting (RBWCF must equal 0). • If the supervisor lock is set (CVTSYLK = 1) and the newly readied task is responsible for it, then the task is dispatchable. If the task is not responsible for the supervisor lock, and it wishes to disable or it is already disabled, it is not a dispatchable task. Otherwise, the task is dispatchable.		SWSETNEW

Diagram 3.17
Dispatcher



1 Determine the next task whose current routine is to be given control.

2 If NEW = OLD, go to Step 6, and give control to the task pointed to by OLD.

3 Save the status of OLD.

4 If the NEW task is indicated, go to Step 6 and give control to NEW.

5 Neither NEW nor OLD is ready. Search for an eligible task.

6 Ensure storage protection.

7 Set the task dispatchable or non-dispatchable.

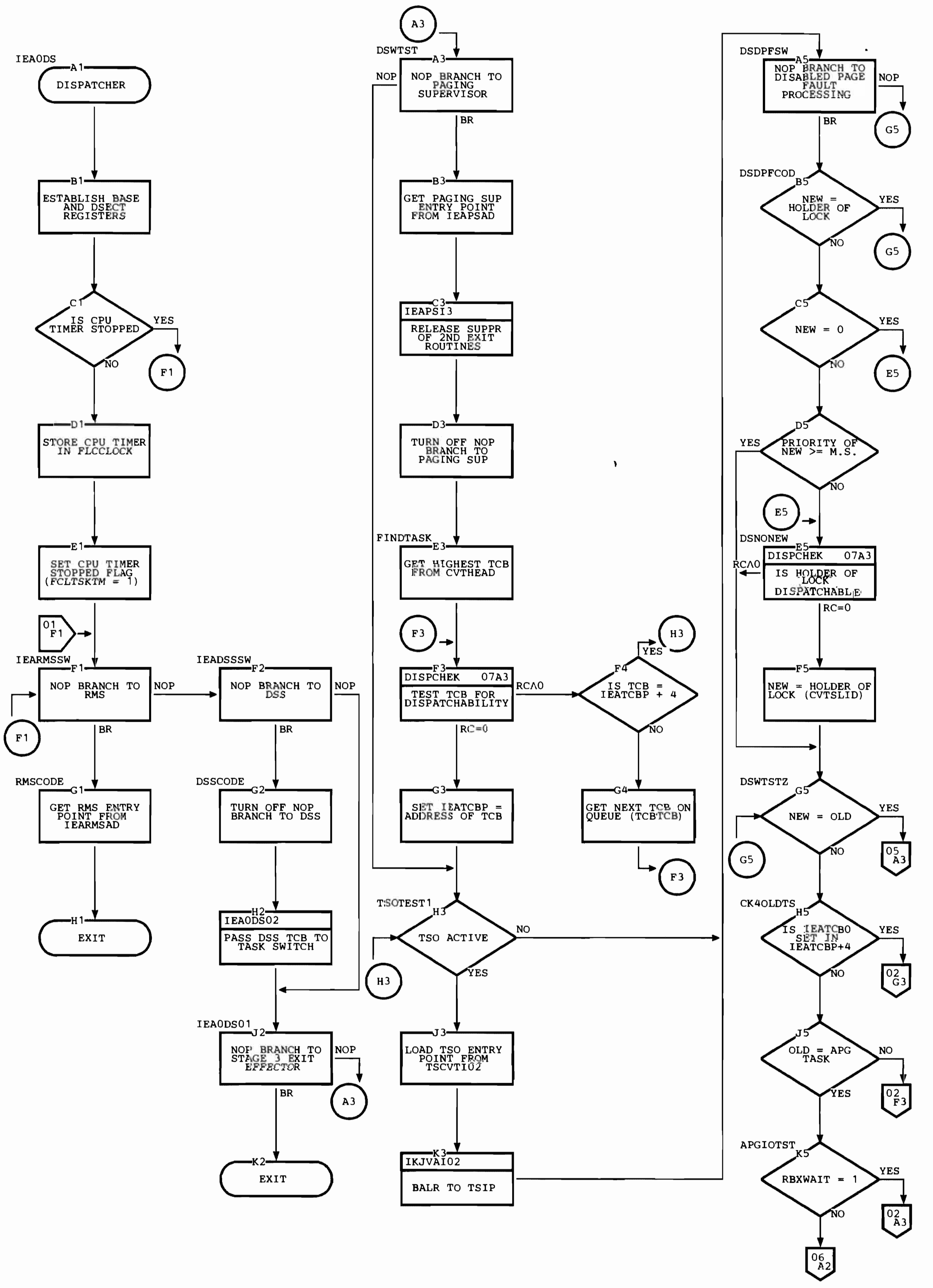
8 Select a region for migration to a secondary paging device.

Diagram 3.17 Dispatcher (Module IEAVNU00)

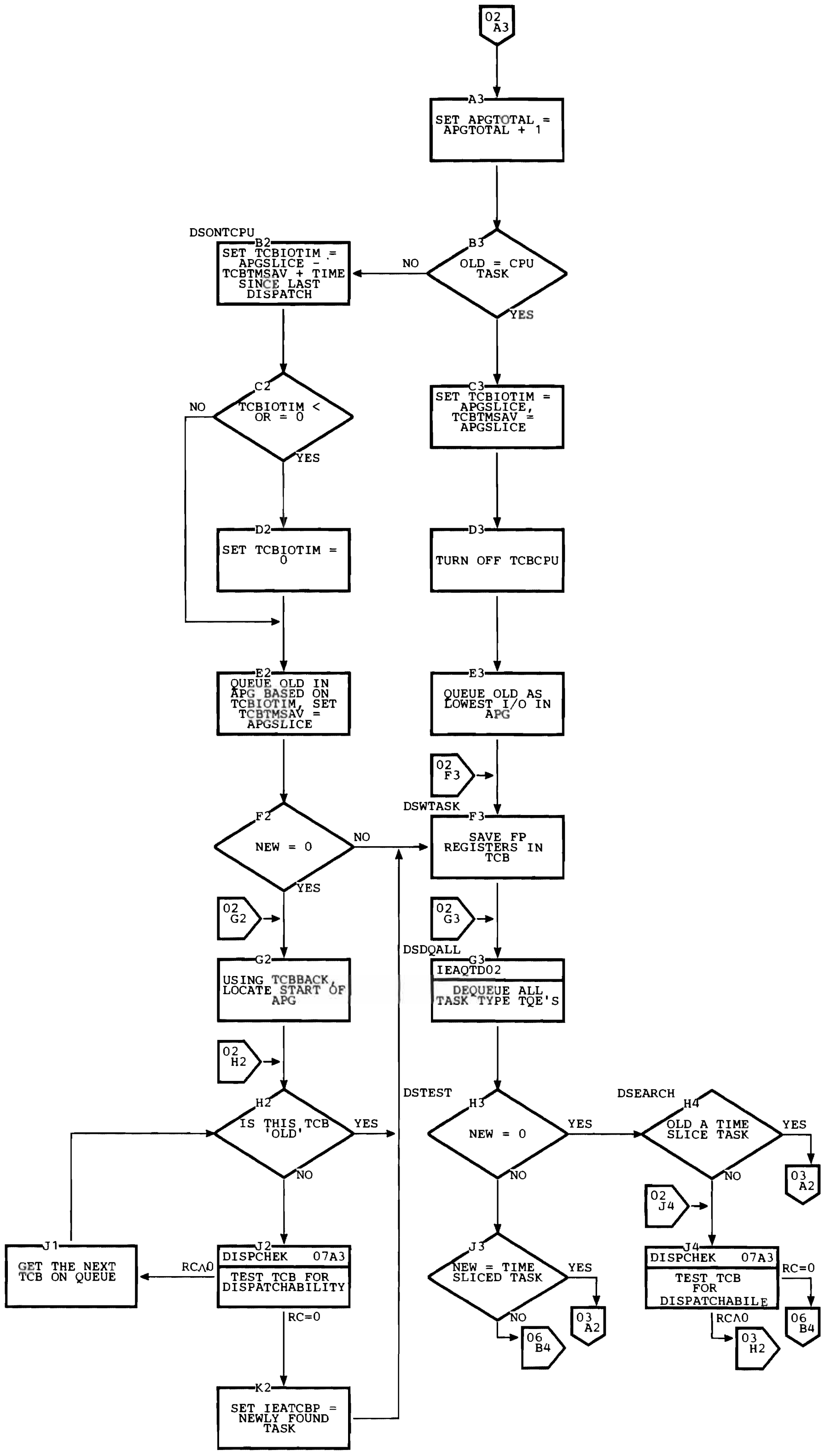
NOTES	ROUTINE NAME	LABEL
Upon entry to the dispatcher, OLD (IEATCBP+4) contains the address of the last user routine's TCB to have control before the supervisor. NEW (IEATCBP) contains the address of the TCB for the program to receive control next, or equals 0, indicating that the dispatcher is to determine which task is to receive control next. If the two TCB pointers are not equal, and the NEW TCB pointer (IEATCBP) does not contain 0, IEATCBP points to the NEW TCB whose current routine will be given control. **1-5** Several possible combinations of the NEW-OLD pointers exist: • NEW equals OLD: It may be possible to redispatch OLD. • NEW equals 0: OLD cannot be dispatched, the task queue should be searched downward from OLD to find a new ready task. • NEW does not equal 0 or OLD: It may be possible to dispatch NEW. • OLD has X'80' in the high-order byte: The OLD was removed from the system before the dispatcher gained control. The task to receive control is determined from the contents of NEW. **2** If OLD=APG then special APG processing is necessary, depending on whether OLD gave up control due to an explicit wait or because its timer interval expired.	Dis-patcher	IEA0DS

NOTES	ROUTINE NAME	LABEL
3 The dispatcher saves registers in the TCB and calls the Timer Dequeue routine to dequeue all task TQEs on the timer queue.	Dis-patcher	DSWTASK
5 The dispatcher calls Timer Enqueue to set up task timing.		DSSEARCH
6 The task that will gain control has been selected. The dispatcher ensures storage protection. The STOR (segment table origin register) must point to the proper segment table. For pageable tasks with a nonzero protection key, segment table entries must be validated.		DSENTER
7 The paging supervisor enters the dispatcher to request that a task be made temporarily nondispatchable. This is an attempt to lower a high paging rate. Nonpageable and TSO tasks are not eligible for selection by the dispatcher. However, if no other eligible task can be found, the dispatcher will call TSO to allow TSO to take action. The dispatcher also sets tasks dispatchable when the paging supervisor requests it.		IEA0DS01
8 Nonpageable regions and TSO regions are not considered for migration. The selected is the job-step task of the lowest dispatching priority among the eligible job steps. No call is made to the dispatcher when pages are being transferred from a secondary paging device to a primary paging device.		

• Chart 3-3 (page 1 of 7). Dispatcher

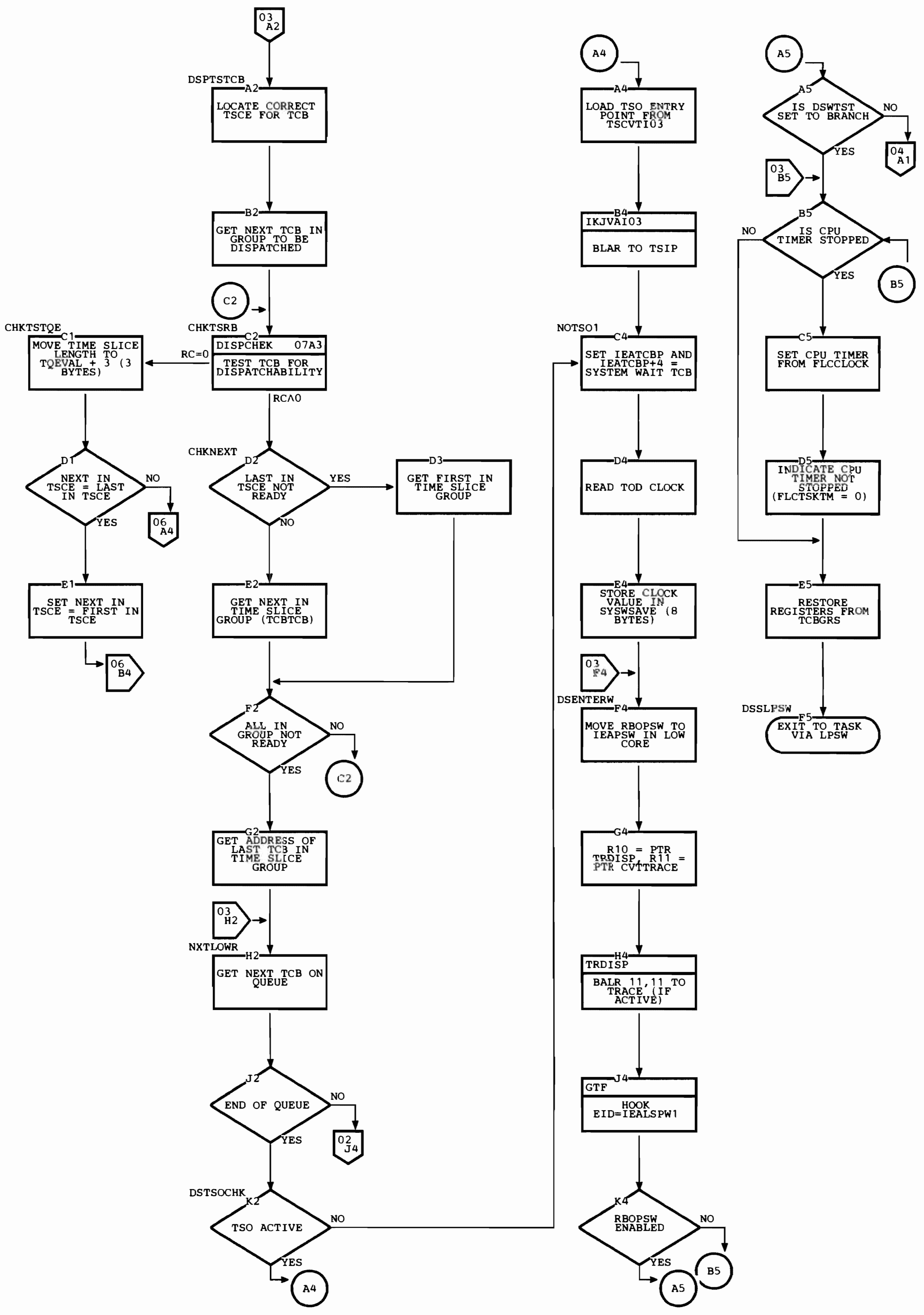

IEAODS A1 DISPATCHER
B1 ESTABLISH BASE AND DSECT REGISTERS
C1 IS CPU TIMER STOPPED
YES
F1
NO
D1 STORE CPU TIMER IN FLCCLOCK
E1 SET CPU TIMER STOPPED FLAG (FCLTSKTM = 1)
01 F1
IEARMSSW F1 NOP BRANCH TO RMS
NOP
BR
F1
RMSCODE G1 GET RMS ENTRY POINT FROM IEARMSAD
H1 EXIT
IEADSSSW F2 NOP BRANCH TO DSS
NOP
BR
DSSCODE G2 TURN OFF NOP BRANCH TO DSS
H2 IEAODS02 PASS DSS TCB TO TASK SWITCH
IEAODS01 J2 NOP BRANCH TO STAGE 3 EXIT EFFECTOR
NOP
A3
BR
K2 EXIT
A3
DSWTST A3 NOP BRANCH TO PAGING SUPERVISOR
NOP
BR
B3 GET PAGING SUP ENTRY POINT FROM IEAPSAD
C3 IEAPSI3 RELEASE SUPPR OF 2ND EXIT ROUTINES
D3 TURN OFF NOP BRANCH TO PAGING SUP
FINDTASK E3 GET HIGHEST TCB FROM CVTHEAD
F3
F3 DISPCHEK 07A3 TEST TCB FOR DISPATCHABILITY
RC≠0
RC=0
G3 SET IEATCBP = ADDRESS OF TCB
F4 IS TCB = IEATCBP + 4
YES
H3
NO
G4 GET NEXT TCB ON QUEUE (TCBTCB)
F3
TSOTEST1 H3 TSO ACTIVE
NO
YES
H3
J3 LOAD TSO ENTRY POINT FROM TSCVTI02
K3 IKJVAI02 BALR TO TSIP
DSDPFSW A5 NOP BRANCH TO DISABLED PAGE FAULT PROCESSING
NOP
G5
BR
DSDPFCOD B5 NEW = HOLDER OF LOCK
YES
G5
NO
C5 NEW = 0
YES
E5
NO
D5 PRIORITY OF NEW >= M.S.
YES
NO
E5
DSNONEW E5 DISPCHEK 07A3 IS HOLDER OF LOCK DISPATCHABLE
RC≠0
RC=0
F5 NEW = HOLDER OF LOCK (CVTSLID)
DSWTSTZ G5 NEW = OLD
YES
05 A3
G5
NO
CK4OLDTS H5 IS IEATCBO SET IN IEATCBP+4
YES
02 G3
NO
J5 OLD = APG TASK
NO
02 F3
YES
APGIOTST K5 RBXWAIT = 1
YES
02 A3
NO
06 A2

• Chart 3-3 (page 2 of 7). Dispatcher



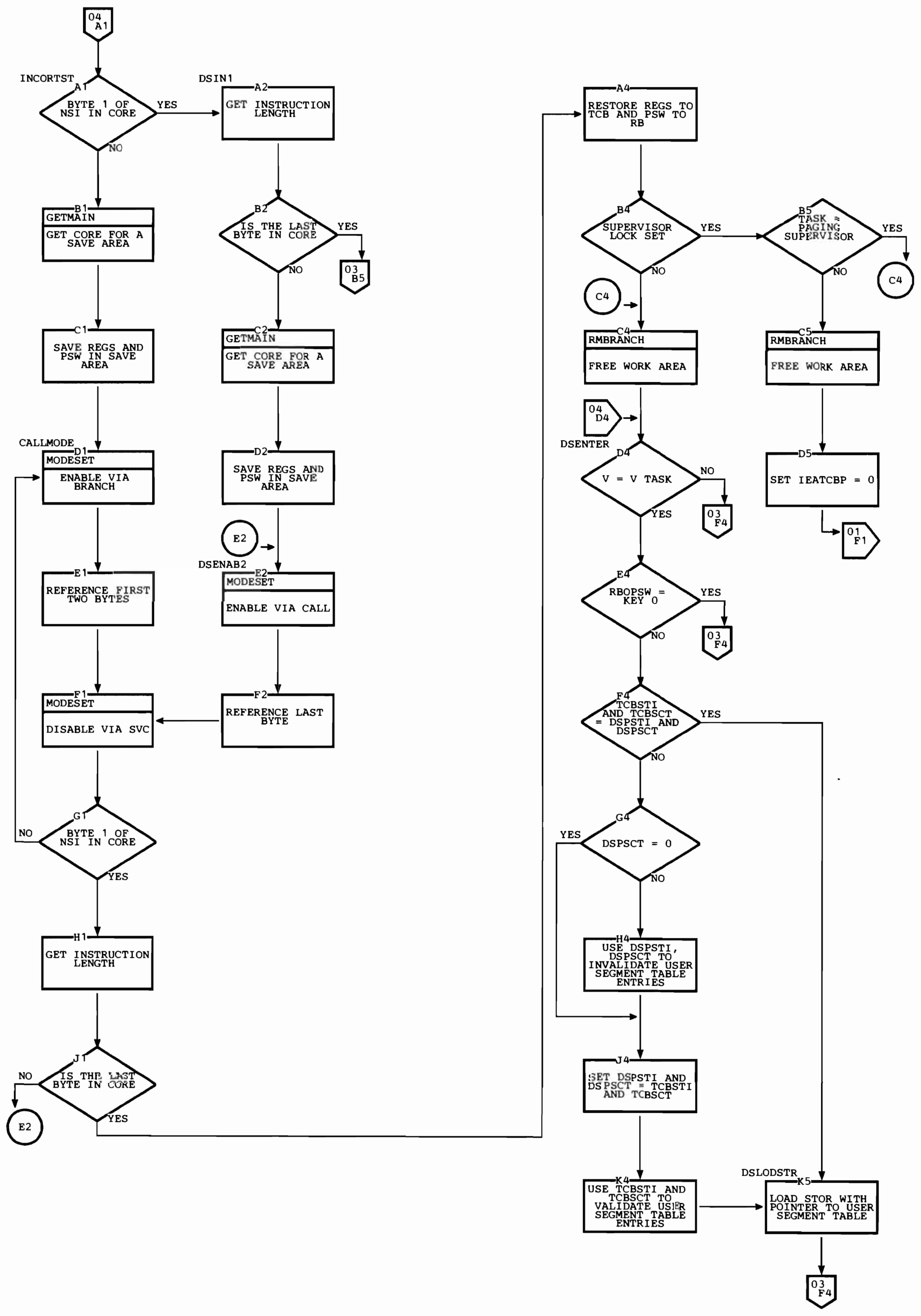
02 A3
A3 SET APGTOTAL = APGTOTAL + 1
B3 OLD = CPU TASK
NO
YES
DSONTCPU
B2 SET TCBIOTIM = APGSLICE - TCBTMSAV + TIME SINCE LAST DISPATCH
C2 TCBIOTIM < OR = 0
NO
YES
C3 SET TCBIOTIM = APGSLICE, TCBTMSAV = APGSLICE
D2 SET TCBIOTIM = 0
D3 TURN OFF TCBCPU
E2 QUEUE OLD IN APG BASED ON TCBIOTIM, SET TCBTMSAV = APGSLICE
E3 QUEUE OLD AS LOWEST I/O IN APG
02 F3
DSWTASK
F3 SAVE FP REGISTERS IN TCB
F2 NEW = 0
NO
YES
02 G2
02 G3
DSDQALL
G2 USING TCBBACK, LOCATE START OF APG
G3 IEAQTD02
DEQUEUE ALL TASK TYPE TQE'S
02 H2
H2 IS THIS TCB 'OLD'
YES
NO
DSTEST
H3 NEW = 0
YES
NO
DSEARCH
H4 OLD A TIME SLICE TASK
YES
NO
03 A2
02 J4
J1 GET THE NEXT TCB ON QUEUE
RC∧0
J2 DISPCHEK 07A3
TEST TCB FOR DISPATCHABILITY
RC=0
J3 NEW = TIME SLICED TASK
YES
NO
03 A2
06 B4
J4 DISPCHEK 07A3
TEST TCB FOR DISPATCHABILE
RC=0
RC∧0
06 B4
03 H2
K2 SET IEATCBP = NEWLY FOUND TASK

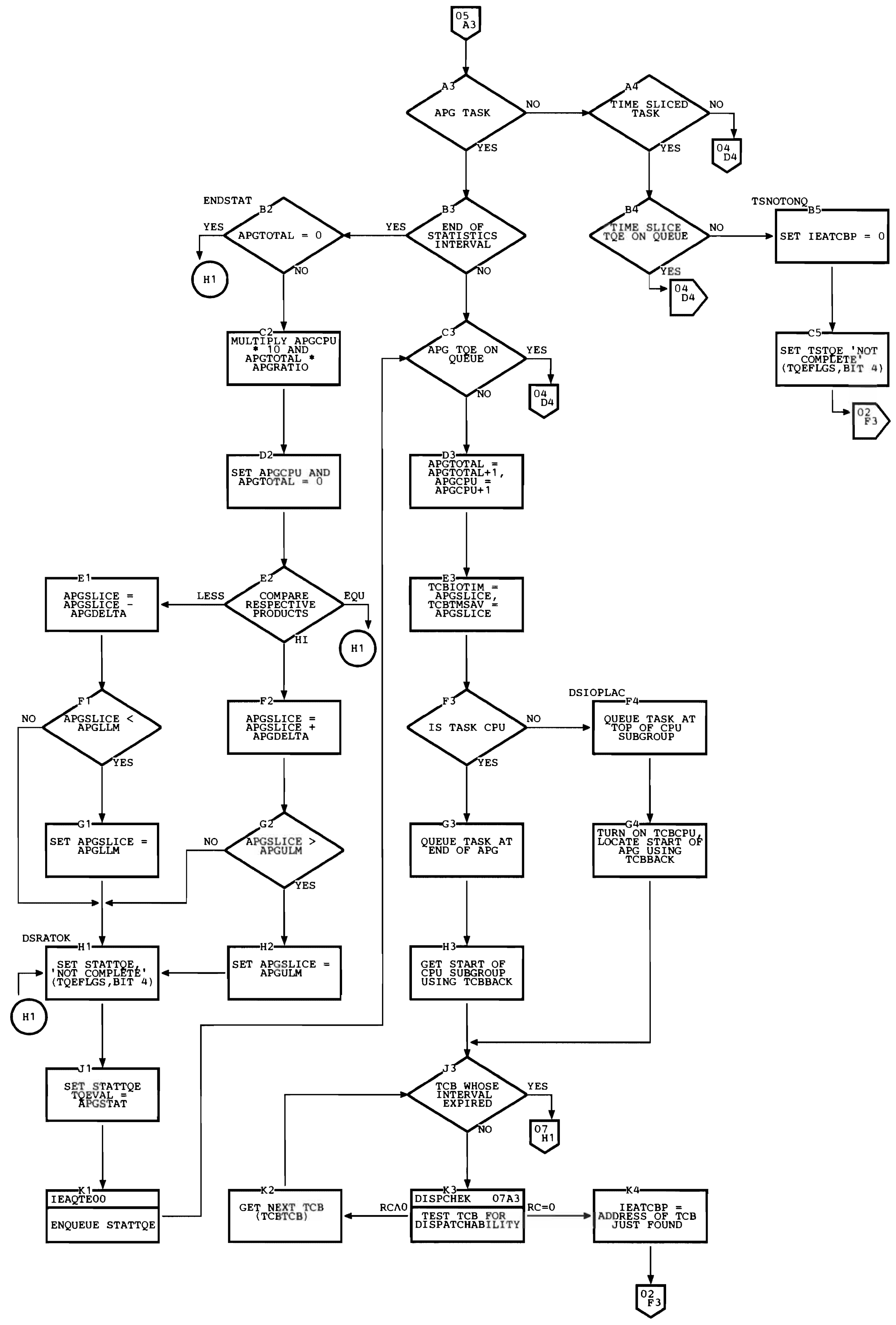
• Chart 3-3 (page 3 of 7). Dispatcher



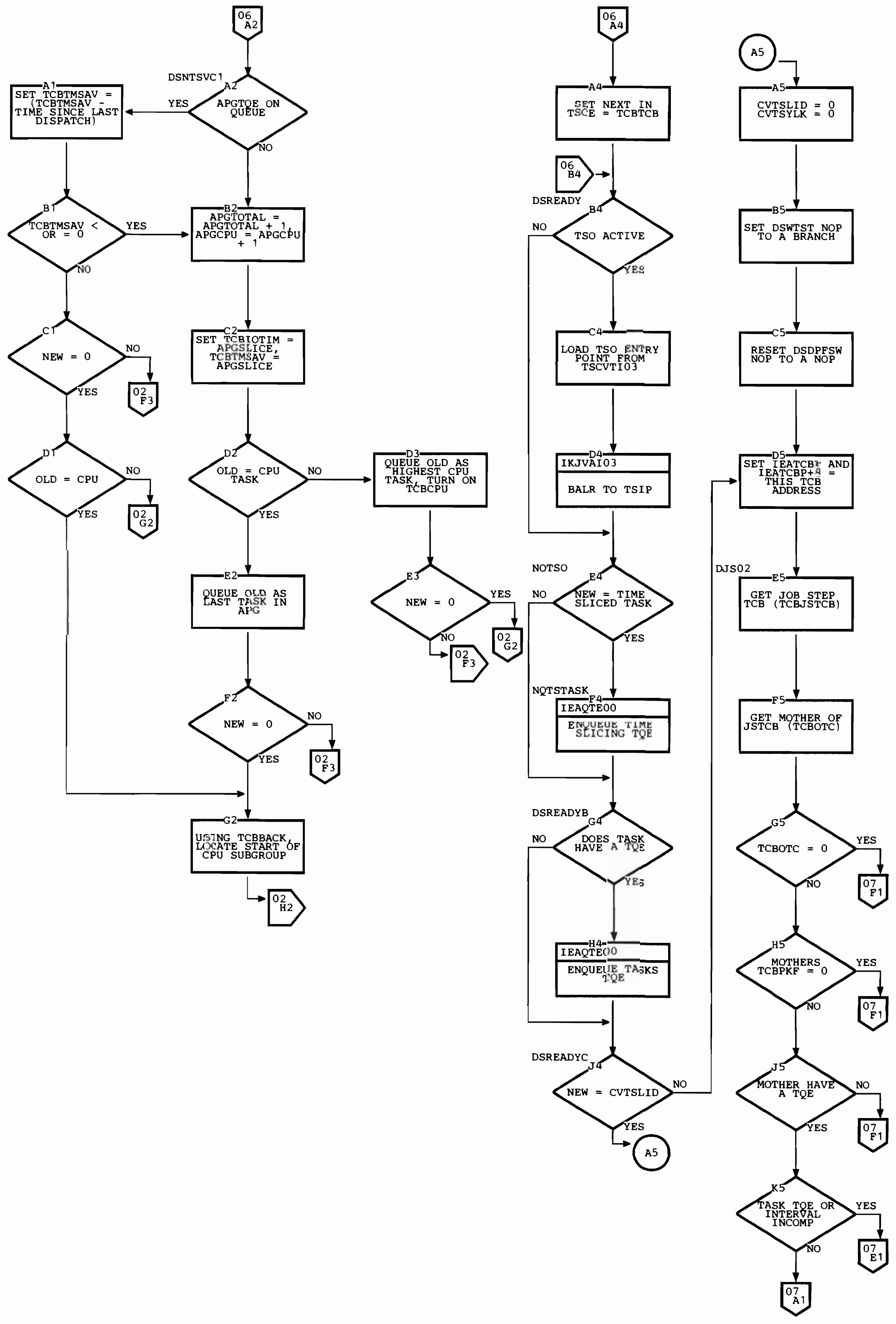
03 A2
DSPTSTCB
A2
LOCATE CORRECT TSCE FOR TCB
B2
GET NEXT TCB IN GROUP TO BE DISPATCHED
C2
CHKTSRB
C2
DISPCHEK 07A3
TEST TCB FOR DISPATCHABILITY
RC=0
RC≠0
CHKTSTQE
C1
MOVE TIME SLICE LENGTH TO TQEVAL + 3 (3 BYTES)
D1
NEXT IN TSCE = LAST IN TSCE
NO
06 A4
YES
E1
SET NEXT IN TSCE = FIRST IN TSCE
06 B4
CHKNEXT
D2
LAST IN TSCE NOT READY
YES
D3
GET FIRST IN TIME SLICE GROUP
NO
E2
GET NEXT IN TIME SLICE GROUP (TCBTCB)
F2
ALL IN GROUP NOT READY
NO
C2
YES
G2
GET ADDRESS OF LAST TCB IN TIME SLICE GROUP
03 H2
NXTLOWR
H2
GET NEXT TCB ON QUEUE
J2
END OF QUEUE
NO
02 J4
YES
DSTSOCHK
K2
TSO ACTIVE
NO
YES
A4
A4
A4
LOAD TSO ENTRY POINT FROM TSCVTI03
B4
IKJVAI03
BLAR TO TSIP
NOTSO1
C4
SET IEATCBP AND IEATCBP+4 = SYSTEM WAIT TCB
D4
READ TOD CLOCK
E4
STORE CLOCK VALUE IN SYSWSAVE (8 BYTES)
03 F4
DSENTERW
F4
MOVE RBOPSW TO IEAPSW IN LOW CORE
G4
R10 = PTR TRDISP, R11 = PTR CVTTRACE
H4
TRDISP
BALR 11,11 TO TRACE (IF ACTIVE)
J4
GTF
HOOK EID=IEALSPW1
K4
RBOPSW ENABLED
NO
B5
YES
A5
A5
A5
IS DSWTST SET TO BRANCH
NO
04 A1
YES
03 B5
B5
IS CPU TIMER STOPPED
NO
YES
B5
C5
SET CPU TIMER FROM FLCCLOCK
D5
INDICATE CPU TIMER NOT STOPPED (FLCTSKTM = 0)
E5
RESTORE REGISTERS FROM TCBGRS
DSSLPSW
F5
EXIT TO TASK VIA LPSW

• Chart 3-3 (page 4 of 7). Dispatcher




05 A3
A3 APG TASK
NO
YES
A4 TIME SLICED TASK
NO
YES
04 D4
ENDSTAT
B2 APGTOTAL = 0
YES
NO
H1
B3 END OF STATISTICS INTERVAL
YES
NO
B4 TIME SLICE TQE ON QUEUE
NO
YES
04 D4
TSNOTONQ
B5 SET IEATCBP = 0
C2 MULTIPLY APGCPU * 10 AND APGTOTAL * APGRATIO
C3 APG TQE ON QUEUE
YES
NO
04 D4
C5 SET TSTQE 'NOT COMPLETE' (TQEFLGS,BIT 4)
02 F3
D2 SET APGCPU AND APGTOTAL = 0
D3 APGTOTAL = APGTOTAL+1, APGCPU = APGCPU+1
E1 APGSLICE = APGSLICE - APGDELTA
LESS
E2 COMPARE RESPECTIVE PRODUCTS
EQU
HI
H1
E3 TCBIOTIM = APGSLICE, TCBTMSAV = APGSLICE
F1 APGSLICE < APGLLM
NO
YES
F2 APGSLICE = APGSLICE + APGDELTA
F3 IS TASK CPU
NO
YES
DSIOPLAC
F4 QUEUE TASK AT TOP OF CPU SUBGROUP
G1 SET APGSLICE = APGLLM
G2 APGSLICE > APGULM
NO
YES
G3 QUEUE TASK AT END OF APG
G4 TURN ON TCBCPU, LOCATE START OF APG USING TCBBACK
DSRATOK
H1 SET STATTQE, 'NOT COMPLETE' (TQEFLGS,BIT 4)
H1
H2 SET APGSLICE = APGULM
H3 GET START OF CPU SUBGROUP USING TCBBACK
J1 SET STATTQE TQEVAL = APGSTAT
J3 TCB WHOSE INTERVAL EXPIRED
YES
NO
07 H1
K1 IEAQTE00
ENQUEUE STATTQE
K2 GET NEXT TCB (TCBTCB)
RC≠0
K3 DISPCHEK 07A3
TEST TCB FOR DISPATCHABILITY
RC=0
K4 IEATCBP = ADDRESS OF TCB JUST FOUND
02 F3

• Chart 3-3 (page 6 of 7). Dispatcher


06 A2
DSNTSVC1
A2 APGTQE ON QUEUE
YES
NO
A1 SET TCBTMSAV = (TCBTMSAV - TIME SINCE LAST DISPATCH)
B1 TCBTMSAV < OR = 0
B2 APGTOTAL = APGTOTAL + 1, APGCPU = APGCPU + 1
C1 NEW = 0
02 F3
C2 SET TCBIOTIM = APGSLICE, TCBTMSAV = APGSLICE
D1 OLD = CPU
02 G2
D2 OLD = CPU TASK
D3 QUEUE OLD AS HIGHEST CPU TASK, TURN ON TCBCPU
E2 QUEUE OLD AS LAST TASK IN APG
E3 NEW = 0
F2 NEW = 0
G2 USING TCBBACK, LOCATE START OF CPU SUBGROUP
02 H2
06 A4
A4 SET NEXT IN TSCE = TCBTCB
06 B4
DSREADY
B4 TSO ACTIVE
C4 LOAD TSO ENTRY POINT FROM TSCVTI03
D4 IKJVAI03 BALR TO TSIP
NOTSO
E4 NEW = TIME SLICED TASK
NQTSTASK
F4 IEAQTE00 ENQUEUE TIME SLICING TQE
DSREADYB
G4 DOES TASK HAVE A TQE
H4 IEAQTE00 ENQUEUE TASKS TQE
DSREADYC
J4 NEW = CVTSLID
A5
A5 CVTSLID = 0 CVTSYLK = 0
B5 SET DSWTST NOP TO A BRANCH
C5 RESET DSDPFSW NOP TO A NOP
D5 SET IEATCB+ AND IEATCBP+4 = THIS TCB ADDRESS
DJS02
E5 GET JOB STEP TCB (TCBJSTCB)
F5 GET MOTHER OF JSTCB (TCBOTC)
G5 TCBOTC = 0
07 F1
H5 MOTHERS TCBPKF = 0
J5 MOTHER HAVE A TQE
K5 TASK TQE OR INTERVAL INCOMP
07 E1
07 A1

• Chart 3-3 (page 7 of 7). Dispatcher

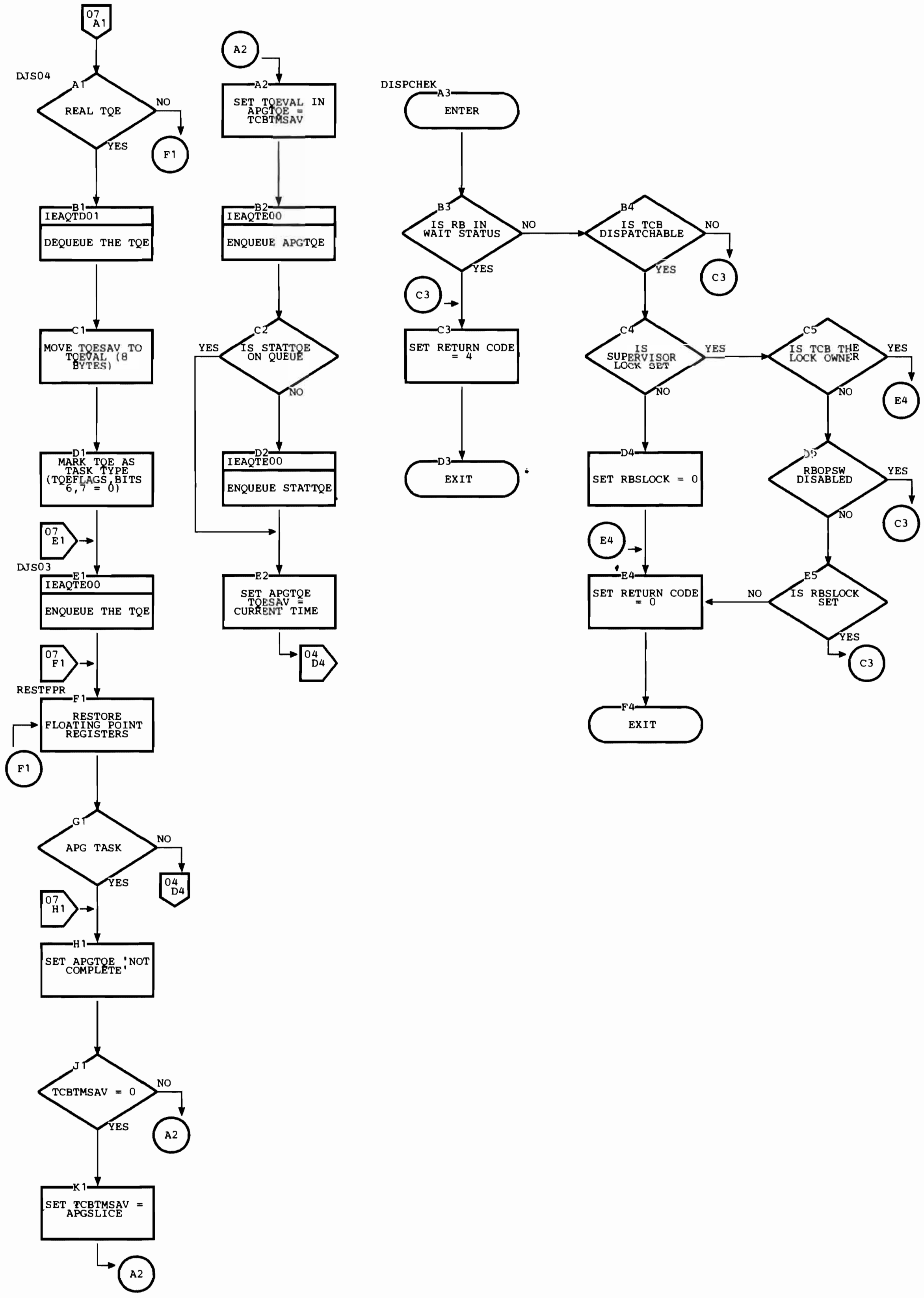
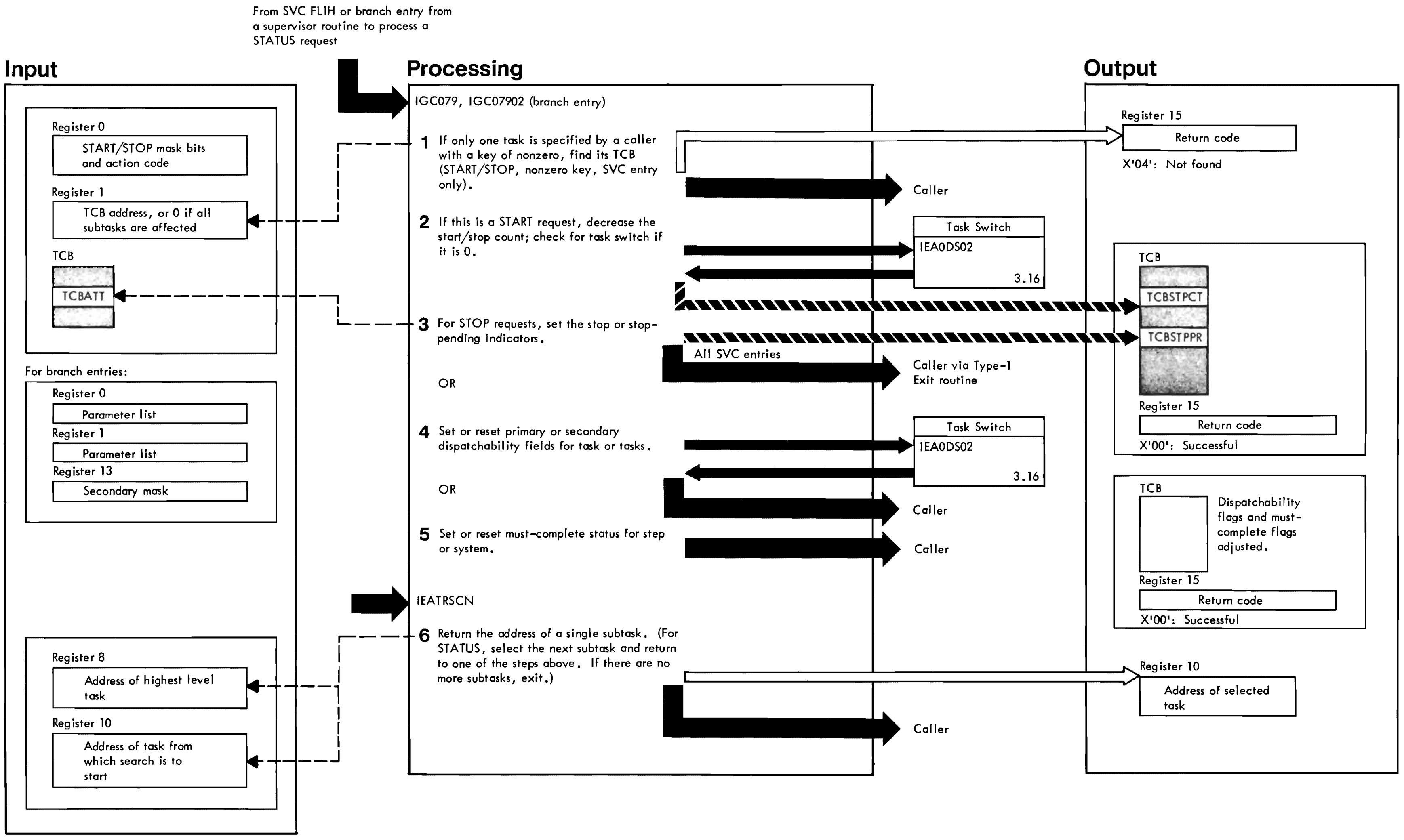


Diagram 3.18
STATUS Routine



From SVC FLIH or branch entry from a supervisor routine to process a STATUS request

Input

Register 0
START/STOP mask bits and action code
Register 1
TCB address, or 0 if all subtasks are affected
TCB
TCBATT

For branch entries:
Register 0
Parameter list
Register 1
Parameter list
Register 13
Secondary mask

Register 8
Address of highest level task
Register 10
Address of task from which search is to start

Processing

IGC079, IGC07902 (branch entry)

1 If only one task is specified by a caller with a key of nonzero, find its TCB (START/STOP, nonzero key, SVC entry only).

2 If this is a START request, decrease the start/stop count; check for task switch if it is 0.

3 For STOP requests, set the stop or stop-pending indicators.

OR

4 Set or reset primary or secondary dispatchability fields for task or tasks.

OR

5 Set or reset must-complete status for step or system.

IEATRSCN

6 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.)

Output

Register 15
Return code
X'04': Not found

TCB
TCBSTPCT
TCBSTPPR
Register 15
Return code
X'00': Successful

TCB
Dispatchability flags and must-complete flags adjusted.
Register 15
Return code
X'00': Successful

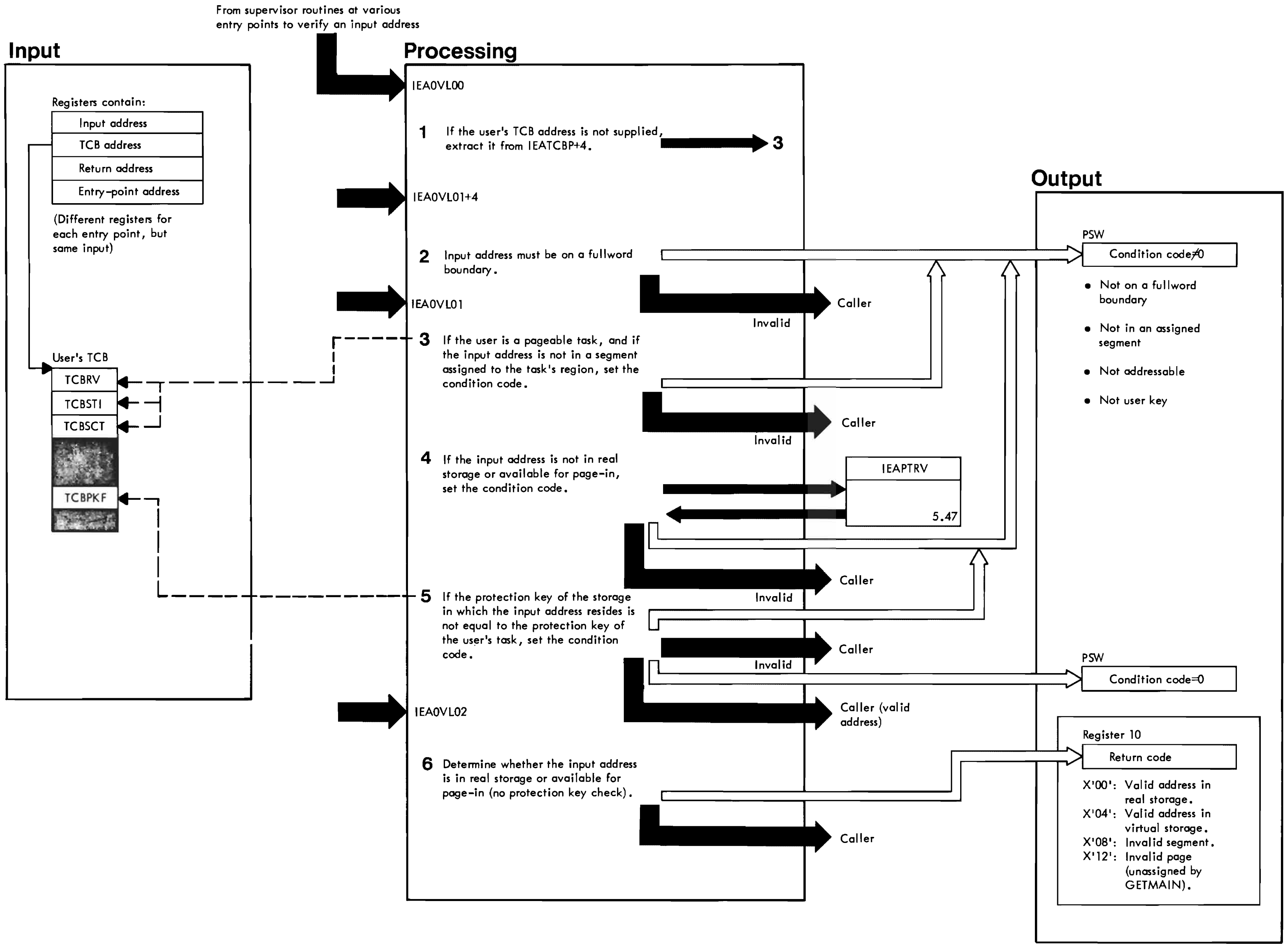
Register 10
Address of selected task

Diagram 3.18 STATUS Routine (Module IEAVSETS)

NOTES	ROUTINE NAME	LABEL
The STATUS routine permits a problem or system program to change TCB fields that control the dispatchability of a task. Routines with a protection key of 0 can use the STATUS routine to set or reset the status of particular tasks. The affected task status can be either the "must-complete" status or the "nondispatchable" status. Problem programs are restricted to the SVC interface, and can control only the start/stop count in the TCB.	STATUS	IGC079
1-3 When a user issues a STATUS macro instruction with the START or STOP operand, the routine determines whether the specified subtask of the current task or all subtasks of the current task are to be modified. When START is specified, the stop/start count is decreased in the subtask TCB(s), and the appropriate dispatchability flags may be cleared. When STOP is specified, the stop/start count is increased in the subtask TCB(s) and the dispatchability flags are set. A task is set nondispatchable only if no system routines are being executed for it, as indicated by the TCBATT flag. If a system routine is being executed for the task, the TCBSTPPR flag is set to indicate that this task should be made nondispatchable when a system routine is no longer being executed for it.		PROCEED
4 The STATUS routine sets the specified dispatchability flag or flags in the specified set of TCBs. (If RESET is specified, the specified dispatchability flag or flags are cleared in the specified set of TCBs.) Four sets of tasks can be specified: the system, the job step, a specified task and its related lower-level tasks, or a single task. If SYSTEM is specified, all tasks of the system are set nondispatchable except the current task and the permanent system tasks. If STEP is specified, all tasks in the job step are set nondispatchable except the current task. If a TCB address starting with the job-step task is specified, the task and its descendants are set nondispatchable. If the current task is among the descendants, its status is not changed. If E is specified, only the task explicitly identified is set nondispatchable. The particular dispatchability flag or flags that are set (or cleared) in each TCB depend on the mask bit number specified in the STATUS macro instruction.		IGC07902

NOTES	ROUTINE NAME	LABEL
6 When entered via the macro instruction STATUS SET, MC, STEP SYSTEM, the STATUS routine sets the caller's task in "step" or "system" must-complete status. (If the RESET operand is specified, the must-complete status that was previously set is cleared.) The routine sets the must-complete flag in the current TCB, the prohibit-asynchronous-exits flag in the current TCB, and the step or system "must-complete" dispatchability flag in other TCBs of the job step or system. If STEP was specified, then all tasks in the job step, including the initiator, are affected. If SYSTEM was specified, then all tasks below the permanent system tasks are affected. For STEP or SYSTEM, the caller's task is always exempt from being set nondispatchable.	STATUS	MCSTEP or MCSYSTEM

• Diagram 3.19 Validity Check Routine


From supervisor routines at various entry points to verify an input address

Input

Registers contain:

- Input address
- TCB address
- Return address
- Entry-point address

(Different registers for each entry point, but same input)

User's TCB

- TCBRV
- TCBSTI
- TCBSCT
- TCBPKF

Processing

IEAOVL00

1 If the user's TCB address is not supplied, extract it from IEATCBP+4. → 3

IEAOVL01+4

2 Input address must be on a fullword boundary.

IEAOVL01

3 If the user is a pageable task, and if the input address is not in a segment assigned to the task's region, set the condition code.

4 If the input address is not in real storage or available for page-in, set the condition code.

5 If the protection key of the storage in which the input address resides is not equal to the protection key of the user's task, set the condition code.

IEAOVL02

6 Determine whether the input address is in real storage or available for page-in (no protection key check).

Output

PSW

Condition code≠0

- Not on a fullword boundary
- Not in an assigned segment
- Not addressable
- Not user key

PSW

Condition code=0

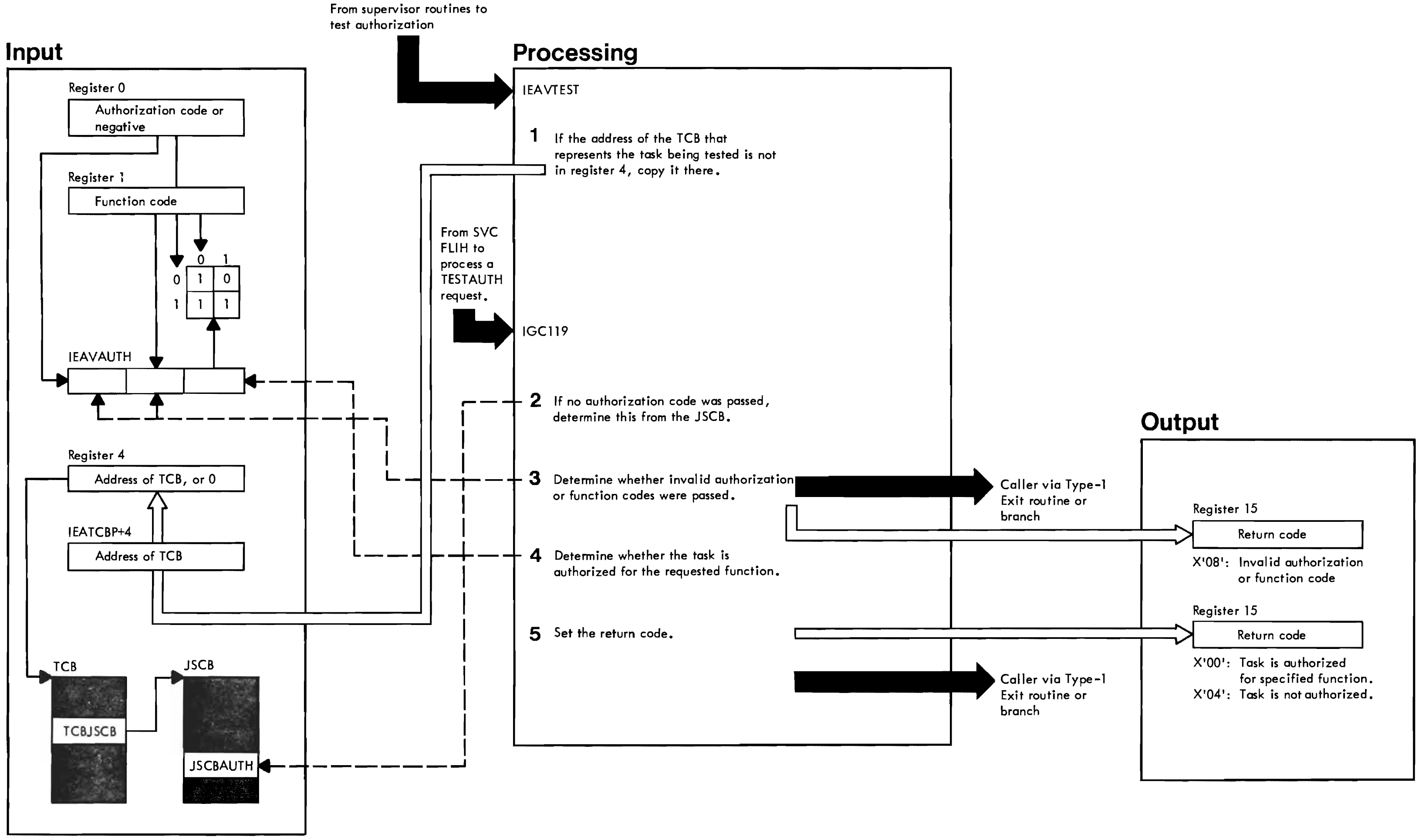
Register 10

Return code

X'00':	Valid address in real storage.
X'04':	Valid address in virtual storage.
X'08':	Invalid segment.
X'12':	Invalid page (unassigned by GETMAIN).

• Diagram 3.19 Validity Check Routine (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
Supervisor routines use the Validity Check routine to check storage addresses passed by user programs. For each input address, the Validity Check routine tests the following: fullword alignment (optional), whether the address is in real storage, whether the address is in a valid segment, and whether the input address is in storage with a protection key that matches the protection key of the caller of the supervisor routine (optional). Because all pageable tasks have the same protection key, Validity Check must perform segment validation to ensure storage security. If an address is invalid, the routine informs the calling supervisor routine by returning a nonzero condition code or a return code. **3** If a user is a pageable task (TCBRV=0), protection key tests are not enough to guarantee system security because all pageable regions have the same protection key. Bits 8-15 of the input address identify the storage segment containing the input address. TCBSTI contains the address of the first segment assigned to the user task, and the TCBSCT field the count of additional segments. The input address must be in the user's storage as defined in the TCB. **4** Validity Check issues an LRA instruction. If the input address is in real storage, or can be paged into real storage, Validity Check simply compares the protection keys in Step 5.	Validity Check	IEAOVL00 IEAOVL01 +4 IEAOVL01
If the return from the LRA instruction indicates that the input address does not point to a valid SGTE (segment table entry), Validity Check sets a nonzero condition code and returns control to the caller. The return from the LRA instruction may indicate that the input address points to a valid SGTE, but that the PTE (page table entry) does not point to a page in real storage. In this case, Validity Check gets the real storage address of the PTE and enters the paging supervisor at IEAPTRV. Subroutine IEAPTRV converts the real storage address of the PTE to a virtual storage address. Validity Check now tests the PGTPAM bit in the PTE. If the bit equals 1, the page containing the input address has been assigned by GETMAIN and is valid. If the page is invalid, Validity Check returns a nonzero condition code. **5** After Validity Check determines whether the input address is within the user's region (Step 4), the storage key of the user's task is compared to the key of the storage in which the input address lies. If the page is not in real storage, the key is found in the XPTPROT field of the XPTE (external page table entry).	Validity Check	
6 Supervisor routines enter Validity Check at IEAOVL02 to determine whether an address is in real storage or available for page-in. The processing is identical to Step 4, except that return codes in register 10 are passed to the calling supervisor routine instead of a condition code in the PSW.		IEAOVL02


From supervisor routines to test authorization
Input
Register 0
Authorization code or negative
Register 1
Function code
0 1
0 1 0
1 1 1
IEAVAUTH
Register 4
Address of TCB, or 0
IEATCBP+4
Address of TCB
TCB
TCBJSCB
JSCB
JSCBAUTH
Processing
IEAVTEST
From SVC FLIH to process a TESTAUTH request.
IGC119
Caller via Type-1 Exit routine or branch
Caller via Type-1 Exit routine or branch
Output
Register 15
Return code
X'08': Invalid authorization or function code
Register 15
Return code
X'00': Task is authorized for specified function.
X'04': Task is not authorized.

1 If the address of the TCB that represents the task being tested is not in register 4, copy it there.

2 If no authorization code was passed, determine this from the JSCB.

3 Determine whether invalid authorization or function codes were passed.

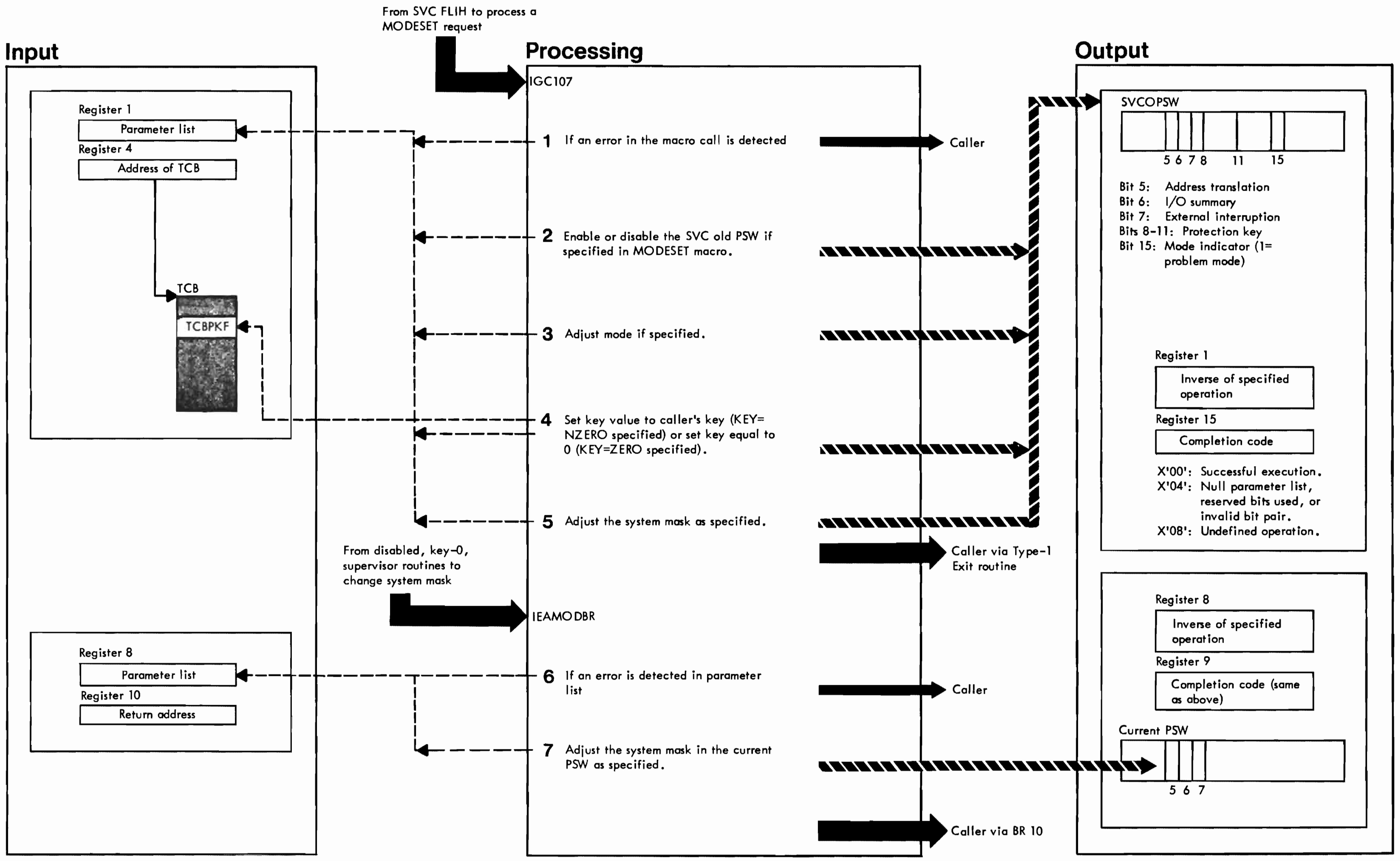
4 Determine whether the task is authorized for the requested function.

5 Set the return code.

Diagram 3.20 TESTAUTH Routine (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
The TESTAUTH routine is called by problem programs or system programs to test whether a task has the authorization to request a specific function. As input parameters, 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.	TESTAUTH	IGC119
3 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.		ROTEST
4 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.		

Diagram 3.21
MODESET Routine



From SVC FLIH to process a MODESET request

Input

Processing

Output

IGC107

1 If an error in the macro call is detected

2 Enable or disable the SVC old PSW if specified in MODESET macro.

3 Adjust mode if specified.

4 Set key value to caller's key (KEY=NZERO specified) or set key equal to 0 (KEY=ZERO specified).

5 Adjust the system mask as specified.

From disabled, key-0, supervisor routines to change system mask

IEAMODBR

6 If an error is detected in parameter list

7 Adjust the system mask in the current PSW as specified.

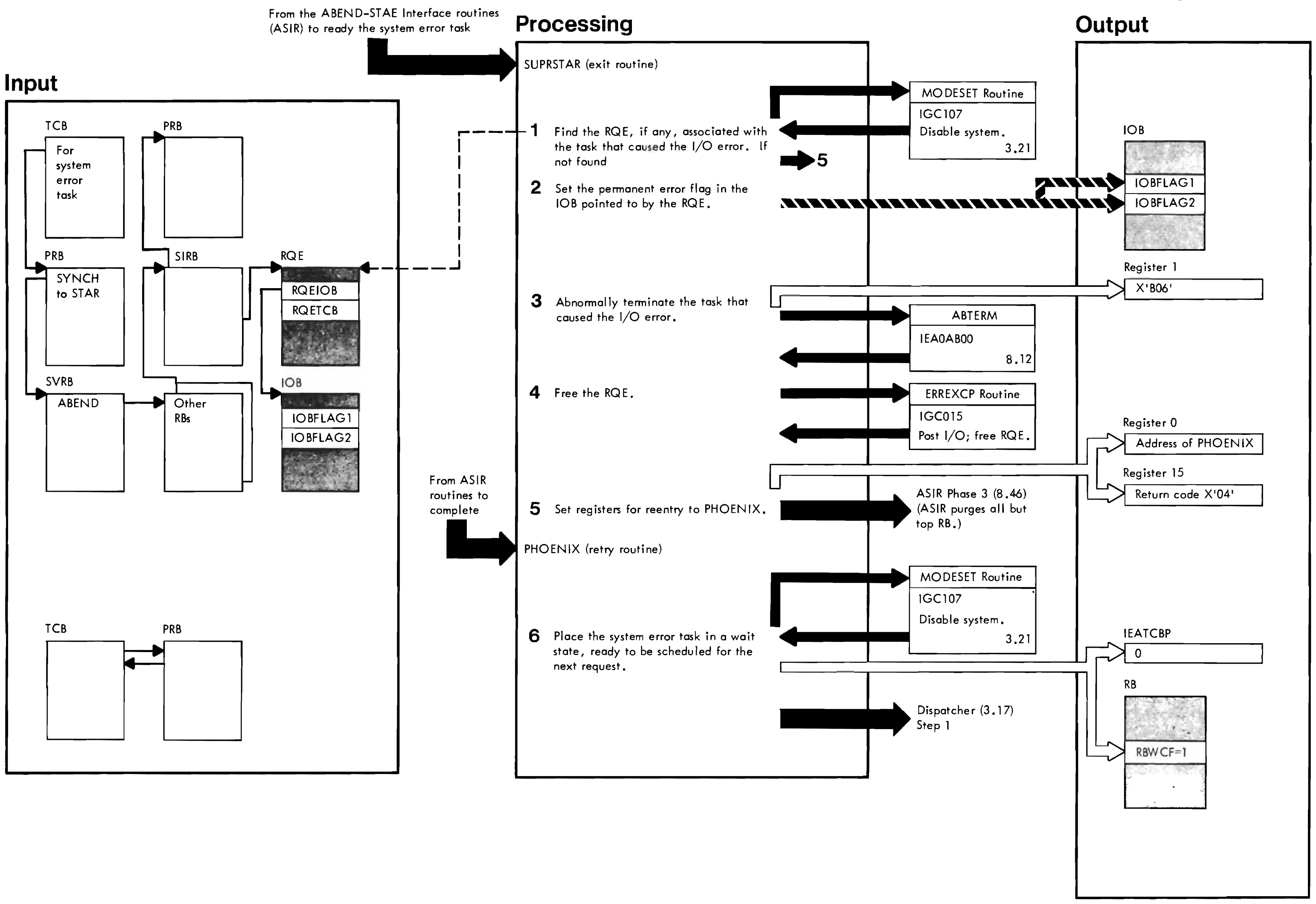
Bit 5: Address translation
Bit 6: I/O summary
Bit 7: External interruption
Bits 8-11: Protection key
Bit 15: Mode indicator (1= problem mode)

X'00': Successful execution.
X'04': Null parameter list, reserved bits used, or invalid bit pair.
X'08': Undefined operation.

Diagram 3.21 MODESET Routine (Module IEAVMODE)

NOTES	ROUTINE NAME	LABEL
By entering the MODESET routine through a macro call, an authorized problem program or system program can change its system mask, change its mode, and change its protection key. In this case, MODESET alters the SVCOPSW, which controls the calling task.	MODESET	IGC107
A disabled supervisor routine with a key of 0 can enter MODESET at a branch entry point to change its own system mask. In this case, MODESET alters the system mask in the current PSW, which belongs to the calling task because no interruption has occurred.		
2 In the MODESET macro, both the ENABLE and SYSMASK operands request changes to the system mask. These two parameters should be coded in a single call to MODESET. MODESET sets the masks requested by the ENABLE operand before setting the masks requested by the SYSMASK operand (Step 5). Therefore, the SYSMASK operand overrides ENABLE when both are coded.		
The operands have slightly different effects. The ENABLE operand specifies that the I/O summary bit (bit 6) and the external interruption bit (bit 7) are to be turned off.		
Through the SYSMASK operand, the caller can individually control bits 5, 6, and 7 of the PSW. Bit 5 is the address translation bit.		
4 If MODESET is setting to a nonzero key, segment validation may be necessary.		
5 When the SYSMASK operand is specified, MODESET places in register 1 the necessary parameters to reestablish the caller's system mask. A special MODESET operand, REG, which cannot be specified in conjunction with any other operands, uses register 1 as set up by MODESET to restore the system mask.		
If SYSMASK is not coded, the contents of register 1 are unpredictable upon return from MODESET.		
6 After a branch entry, MODESET places the parameters necessary to restore the caller's system mask in register 8.		

Diagram 3.22
System Task ABEND Recovery (STAR) Exit Routine of the System Error Task



SUPRSTAR (exit routine)

1 Find the RQE, if any, associated with the task that caused the I/O error. If not found ➡5

2 Set the permanent error flag in the IOB pointed to by the RQE.

3 Abnormally terminate the task that caused the I/O error.

4 Free the RQE.

5 Set registers for reentry to PHOENIX.

PHOENIX (retry routine)

6 Place the system error task in a wait state, ready to be scheduled for the next request.

Diagram 3.22 System Task ABEND Recovery (STAR) Exit Routine of the System Error Task (Module IEAVNU00)

NOTES	ROUTINE NAME	LABEL
When the system error task fails, the System Task ABEND Recovery (STAR) routine gains control. The STAR routine is in the system error task, and has two parts -- an exit routine and a retry routine. Both the exit and retry routines are disabled during most of their execution. The exit routine schedules abnormal termination of the task that the system error task was trying to correct. The retry routine places the system error task in a wait state, ready to service another I/O error.	STAR	
1 The STAR routine searches the RB queue of the system error task for the SIRB, trying to find the RQE (request queue element) that belongs to the task that caused the I/O error. RQEs are built by data management routines and are used by the I/O supervisor to schedule I/O requests. If STAR finds an RQE, it inspects the RQETCB field to determine whether this is a normal RQE (RQETCB ≠ 0), or a dummy RQE (RQETCB = 0). If the RQE is a dummy, STAR records a paging device error, and there is no task to abnormally terminate. In this case, STAR continues at Step 5. If the RQE points to an IOB (input/output block), STAR turns off the error-in-process flag (IOBFLAG1), and sets the permanent-error flag (IOBFLAG2).		SUPRSTAR
6	STAR	PHOENIX

SECTION 4

Contents Supervision

4

The routines of contents supervision search virtual and auxiliary storage for requested modules and schedule their use. The LINK, SYNCH, LOAD, XCTL, DELETE, and IDENTIFY SVC routines comprise contents supervision. The overlay supervisor and Program Fetch are also part of contents supervision.

Diagram 4.0 (a visual table of contents to the diagrams in this section) and Diagram 4.1 (a diagram showing the overall logic of contents supervision) point to the diagrams for each of the above routines. To help you understand the operations described by the diagrams, the following subsections describe:

- Calling the LINK, LOAD, and XCTL routines by issuing a macro instruction (see _OS/VS Supervisor Services and Macro Instructions_ for detailed information).

- Searching virtual and auxiliary storage to find requested modules.

CALLING THE LINK, LOAD, AND XCTL ROUTINES

Problem programs and system programs call contents supervision to find a requested module by issuing a LINK, LOAD, or XCTL macro instruction. Certain important operands in the macro call determine which storage is searched and the order of the search. The operands are shown in Figure 4-1. Only one of the first three operands can be coded, and the DCB operand is optional.

Operand	Meaning
EP	is the entry-point name in the module to be given control.
EPLOC	is the address of the entry-point name in the module to be given control.
DE	is the name field of a list entry for the entry-point name. The list entry is constructed using the BLDL macro instruction. The DCB operand must indicate the same DCB (data control block) used in the BLDL macro instruction.
DCB	is the address of the DCB for the partitioned data set containing the entry-point name for the module to be given control. The address of the data control block for the link and job libraries is designated by specifying an address of 0, or by omitting the DCB operand.

Figure 4-1. Important operands in the LINK, LOAD, and XCTL macro instructions.

SEARCHING VIRTUAL AND AUXILIARY STORAGE

The contents supervision routines find a module by scanning control blocks that represent modules. These control blocks form different queues and directories, and each queue or directory describes a different part of storage. The queues and directories are:

- JPA (job pack area)

- LPA (link pack area)

- TSLPA (time sharing link pack area)

- Auxiliary storage libraries

The Job Pack Area

The JPA (job pack area) in virtual storage contains modules needed for the execution of jobs. The JPA is 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.

Modules in the JPA are represented by CDEs (contents directory entries). Each CDE contains:

- The name of the module it represents.

- A pointer to the module's entry point.

- A use count (in fields CDROLL and CDUSE) which represents the total number of successful requests for a module by ATTACH, LINK, LOAD, and XCTL macro instructions. (The maximum use count is 4095.)

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, and 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, and the first CDE on the JPAQ represents the first module requested by the job step. The beginning of the JPAQ is pointed to by the TCBJPQ field in the job-step TCB.

The Load List: Each time a module is allocated to a requester by the LOAD routine, the use count in the CDE is increased, as noted previously. 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 searched by the LOAD routine. Figure 4-2 shows the control blocks for modules in the JPA, including LLEs.

The need for a responsibility count in the LLE separate from the use count in the CDE is not readily apparent. Each time a module is successfully allocated by the LOAD routine, 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 areas if they are both 0, meaning that there are no more outstanding requests.

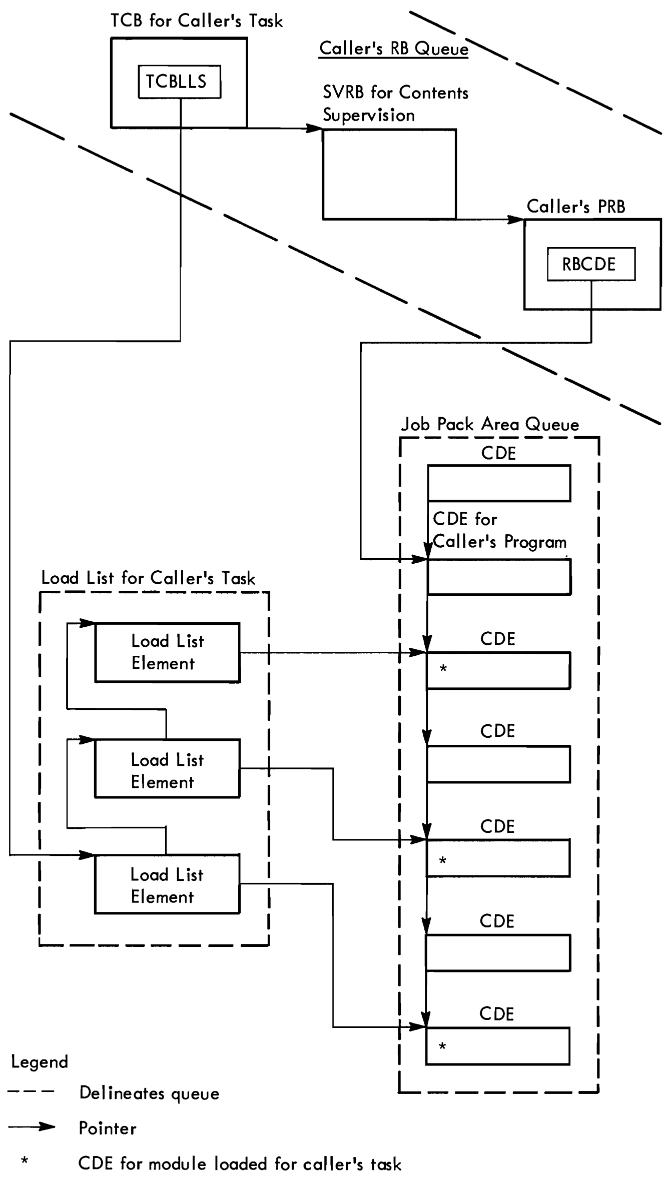


Figure 4-2. Control blocks for modules in the JPA.

LPA Storage Areas

The LPA (link pack area) contains: (1) modules that may be executed concurrently by all tasks in the system, and (2) a directory of those modules. There are three LPA storage areas:

- The LPA directory
- The link pack area
- The Fixed LPA

The LPA Directory: The LPA directory is created in pageable storage during nucleus initialization. It contains an LPDE (link pack directory entry) for each entry point in the LPA modules. LPDEs for major entry points contain a CDE and a compressed extent list, while LPDEs for alias entry points contain the name of a related major LPDE rather than a compressed extent list. LPDEs are organized during nucleus initialization, and are not chained as are the CDEs on the LPAQ and the JPAQ. Diagram 4.4 describes the search for an LPDE that represents a module that cannot be found on the LPAQ.

The Link Pack Area: The link pack area contains the LPA modules. All type-3 and type-4 SVC routines and all ERPs (error recovery procedures) are placed in this area, which is pageable.

All LPA modules in use are represented by CDEs on the LPAQ (link pack area queue). When an LPA module is no longer needed (the use count in the CDE reached zero), the control blocks that represent it in the LPA are removed by the Exit routine. These modules are still represented by LPDEs on the LPA directory.

The Fixed LPA: During nucleus initialization, the user may specify that any module in the LPA is placed in a permanently fixed area of real storage. This area is called the fixed LPA and is not paged. These modules are also represented by CDEs on the LPAQ, and they are used in preference to an identical paged copy of the module in the LPA.

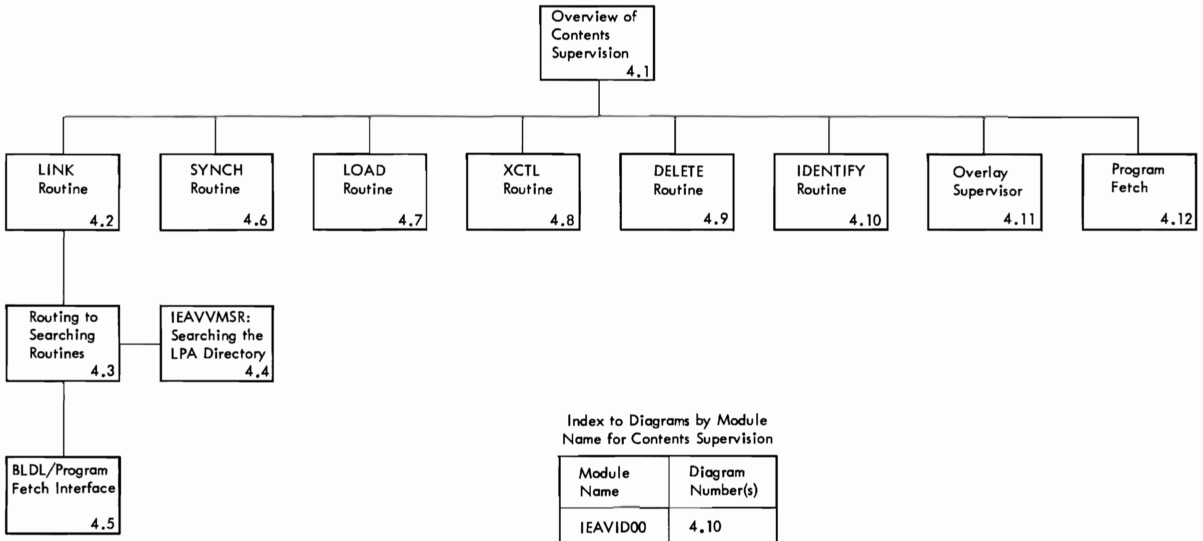
Time Sharing Link Pack Area

The TSLPA (time sharing link pack area) is in the TSC (time sharing control) region, and contains TSO modules that may be executed concurrently by all TSO tasks in the system. These modules are represented by CDEs on the TSLPAQ (time sharing link pack area queue).

Auxiliary Storage Libraries

When the contents supervision routines cannct find a requested module in virtual storage, BLDL searches the libraries on auxiliary storage. Modules on auxiliary storage are represented by PDS DEs (partitioned data set directory entries).

• Diagram 4.0 Contents Supervision Visual Table of Contents



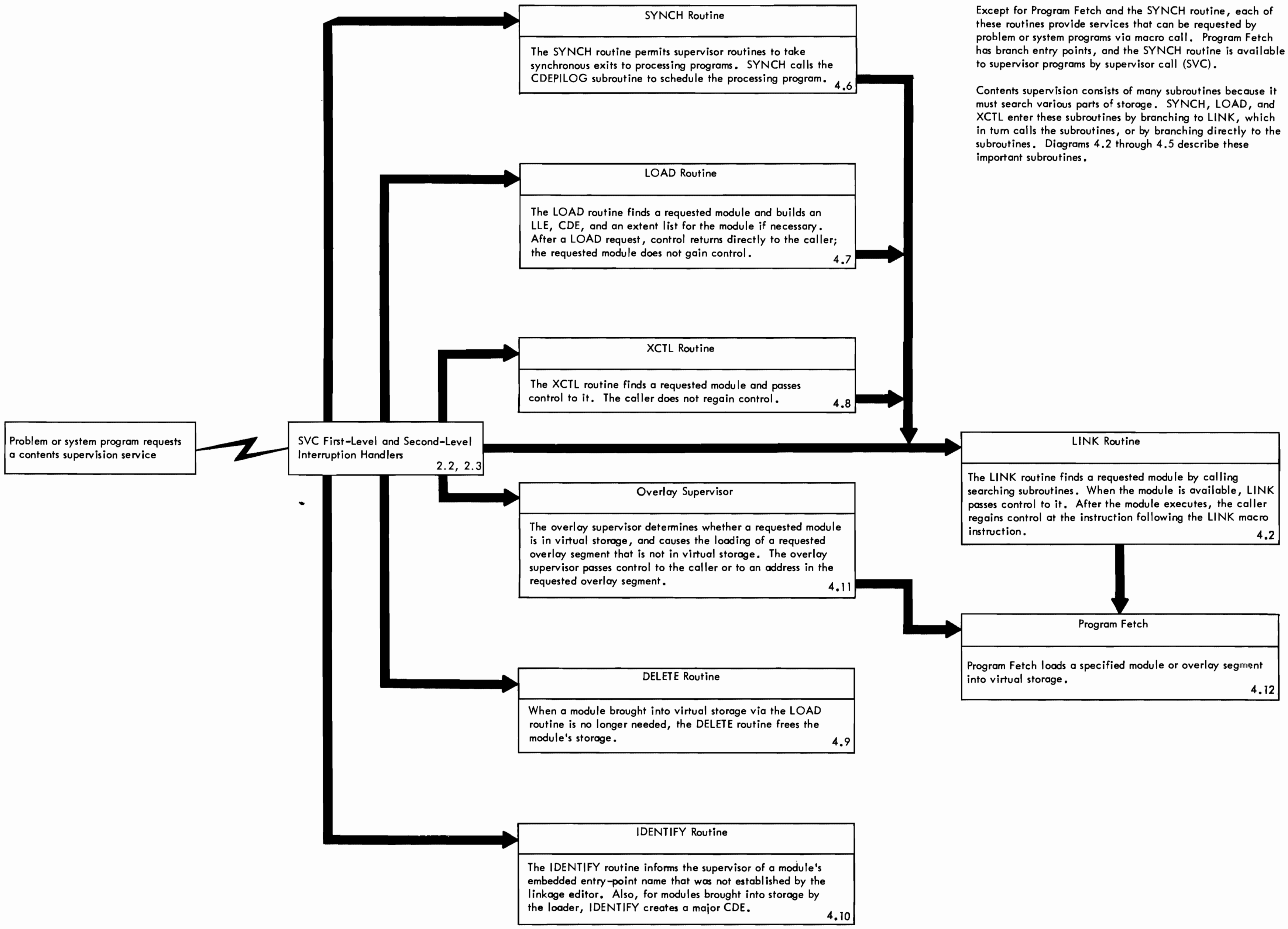
Index to Diagrams by Module Name for Contents Supervision

Module Name	Diagram Number(s)
IEAVID00	4.10
IEAVLK00	4.2, 4.4, 4.6--4.9
IEAVLK01	4.3, 4.5
IEWFETCH	4.12
IEWSUOVR	4.11
IEWSWOVR	4.11

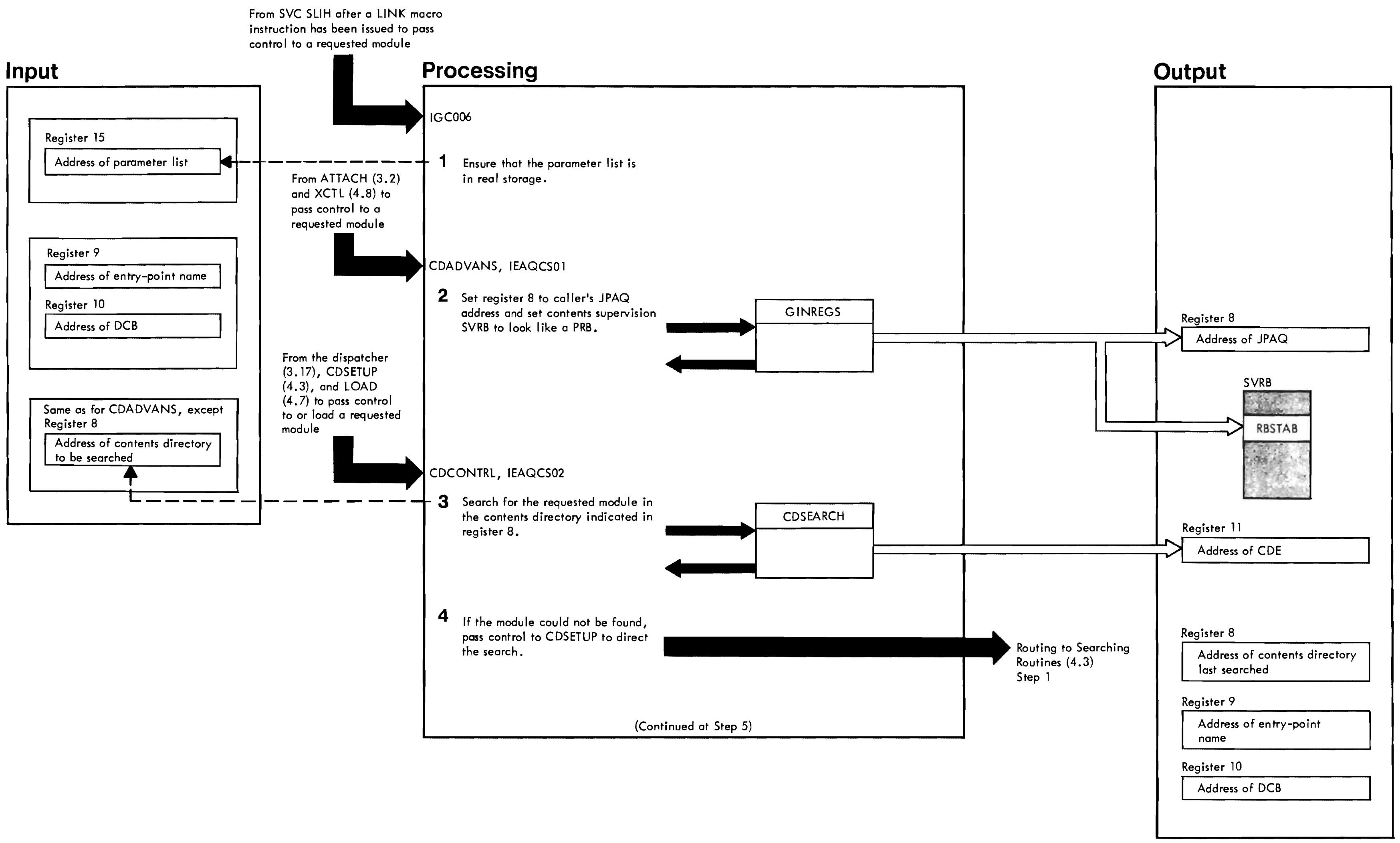
• **Diagram 4.1**
Overview of Contents Supervision

Except for Program Fetch and the SYNCH routine, each of these routines provide services that can be requested by problem or system programs via macro call. Program Fetch has branch entry points, and the SYNCH routine is available to supervisor programs by supervisor call (SVC).

Contents supervision consists of many subroutines because it must search various parts of storage. SYNCH, LOAD, and XCTL enter these subroutines by branching to LINK, which in turn calls the subroutines, or by branching directly to the subroutines. Diagrams 4.2 through 4.5 describe these important subroutines.



Problem or system program requests a contents supervision service
SVC First-Level and Second-Level Interruption Handlers 2.2, 2.3
SYNCH Routine
The SYNCH routine permits supervisor routines to take synchronous exits to processing programs. SYNCH calls the CDEPILOG subroutine to schedule the processing program. 4.6
LOAD Routine
The LOAD routine finds a requested module and builds an LLE, CDE, and an extent list for the module if necessary. After a LOAD request, control returns directly to the caller; the requested module does not gain control. 4.7
XCTL Routine
The XCTL routine finds a requested module and passes control to it. The caller does not regain control. 4.8
Overlay Supervisor
The overlay supervisor determines whether a requested module is in virtual storage, and causes the loading of a requested overlay segment that is not in virtual storage. The overlay supervisor passes control to the caller or to an address in the requested overlay segment. 4.11
DELETE Routine
When a module brought into virtual storage via the LOAD routine is no longer needed, the DELETE routine frees the module's storage. 4.9
IDENTIFY Routine
The IDENTIFY routine informs the supervisor of a module's embedded entry-point name that was not established by the linkage editor. Also, for modules brought into storage by the loader, IDENTIFY creates a major CDE. 4.10
LINK Routine
The LINK routine finds a requested module by calling searching subroutines. When the module is available, LINK passes control to it. After the module executes, the caller regains control at the instruction following the LINK macro instruction. 4.2
Program Fetch
Program Fetch loads a specified module or overlay segment into virtual storage. 4.12



From SVC SLIH after a LINK macro instruction has been issued to pass control to a requested module

Input

Register 15
- Address of parameter list

Register 9
- Address of entry-point name

Register 10
- Address of DCB

Same as for CDADVANS, except Register 8
- Address of contents directory to be searched

Processing

IGC006

1 Ensure that the parameter list is in real storage.

From ATTACH (3.2) and XCTL (4.8) to pass control to a requested module

CDADVANS, IEAQCS01

2 Set register 8 to caller's JPAQ address and set contents supervision SVRB to look like a PRB.

From the dispatcher (3.17), CDSETUP (4.3), and LOAD (4.7) to pass control to or load a requested module

CDCONTRL, IEAQCS02

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.

(Continued at Step 5)

Output

Register 8
- Address of JPAQ

Register 11
- Address of CDE

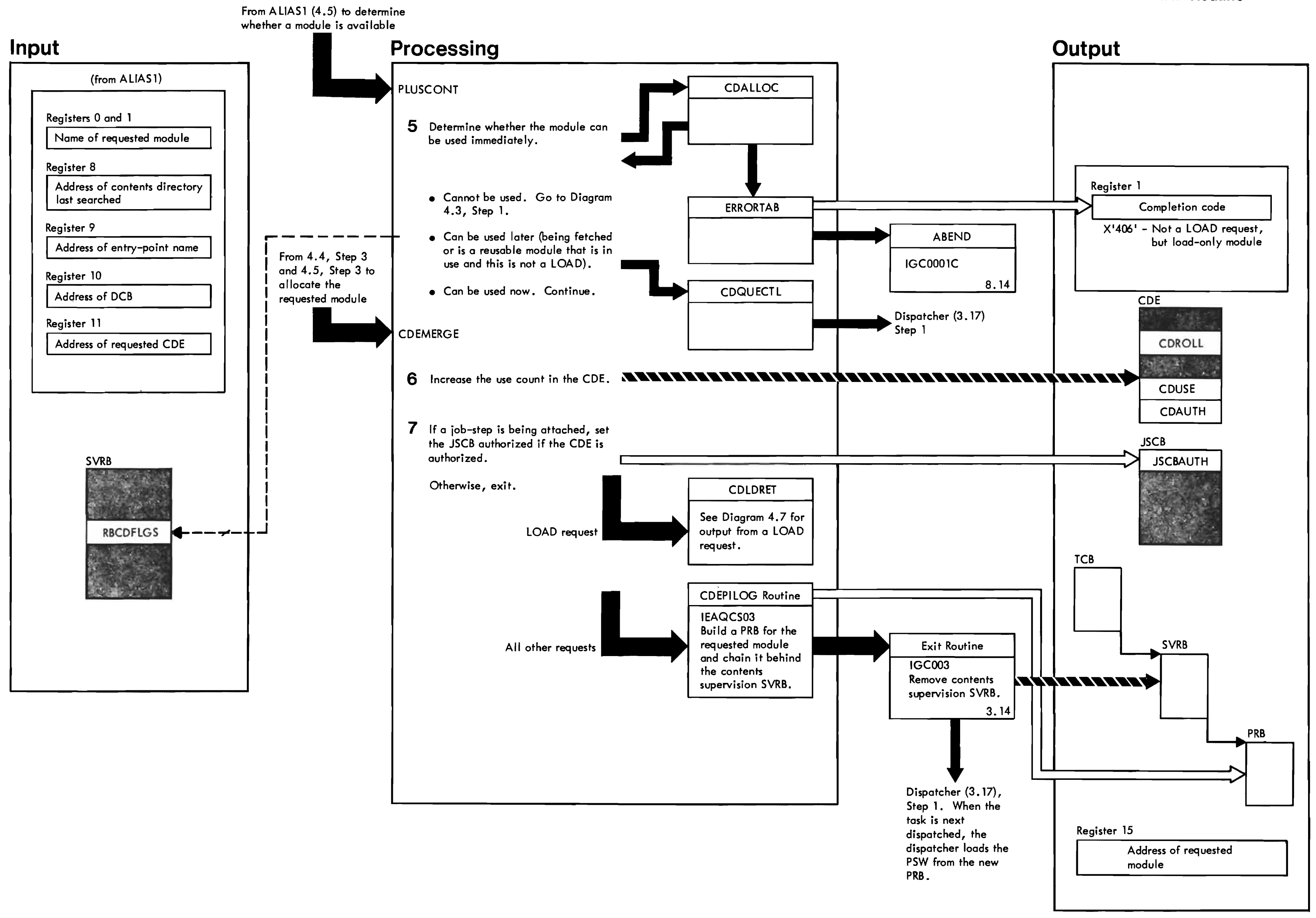
Register 8
- Address of contents directory last searched

Register 9
- Address of entry-point name

Register 10
- Address of DCB

• Diagram 4.2 (Steps 1-4) LINK Routine (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
The LINK routine is the central routine of contents supervision. It directs the XCTL, LOAD, and ATTACH routines to the subroutines of contents supervision to find and obtain linkage to a requested module, or to load a requested module.	LINK	IGC006
1 LINK calls LXPREFIX to ensure that the parameter list is in real storage.	LXPREFIX	LXPREFIX
2 The two high-order bits of the RESTAB field in the SVRB (supervisor request block) are set to 1 to make the SVRB look like a PRB (program request block). Now ABEND can free the module if the requester abnormally terminates. The address of the requester's JPAQ (jcb pack area queue) is placed in register 8 to indicate to CDSEARCH which queue is to be searched.	GINREGS	GINREGS


From ALIAS1 (4.5) to determine whether a module is available
Input
(from ALIAS1)
Registers 0 and 1
Name of requested module
Register 8
Address of contents directory last searched
Register 9
Address of entry-point name
Register 10
Address of DCB
Register 11
Address of requested CDE
SVRB
RBCDFLGS
From 4.4, Step 3 and 4.5, Step 3 to allocate the requested module
Processing
PLUSCONT
CDEMERGE
CDALLOC
ERRORTAB
ABEND
IGC0001C
8.14
CDQUECTL
Dispatcher (3.17) Step 1
LOAD request
CDLDRET
See Diagram 4.7 for output from a LOAD request.
All other requests
CDEPILOG Routine
IEAQCS03
Build a PRB for the requested module and chain it behind the contents supervision SVRB.
Exit Routine
IGC003
Remove contents supervision SVRB.
3.14
Dispatcher (3.17), Step 1. When the task is next dispatched, the dispatcher loads the PSW from the new PRB.
Output
Register 1
Completion code
X'406' - Not a LOAD request, but load-only module
CDE
CDROLL
CDUSE
CDAUTH
JSCB
JSCBAUTH
TCB
SVRB
PRB
Register 15
Address of requested module

5 Determine whether the module can be used immediately.

- Cannot be used. Go to Diagram 4.3, 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.

6 Increase the use count in the CDE.

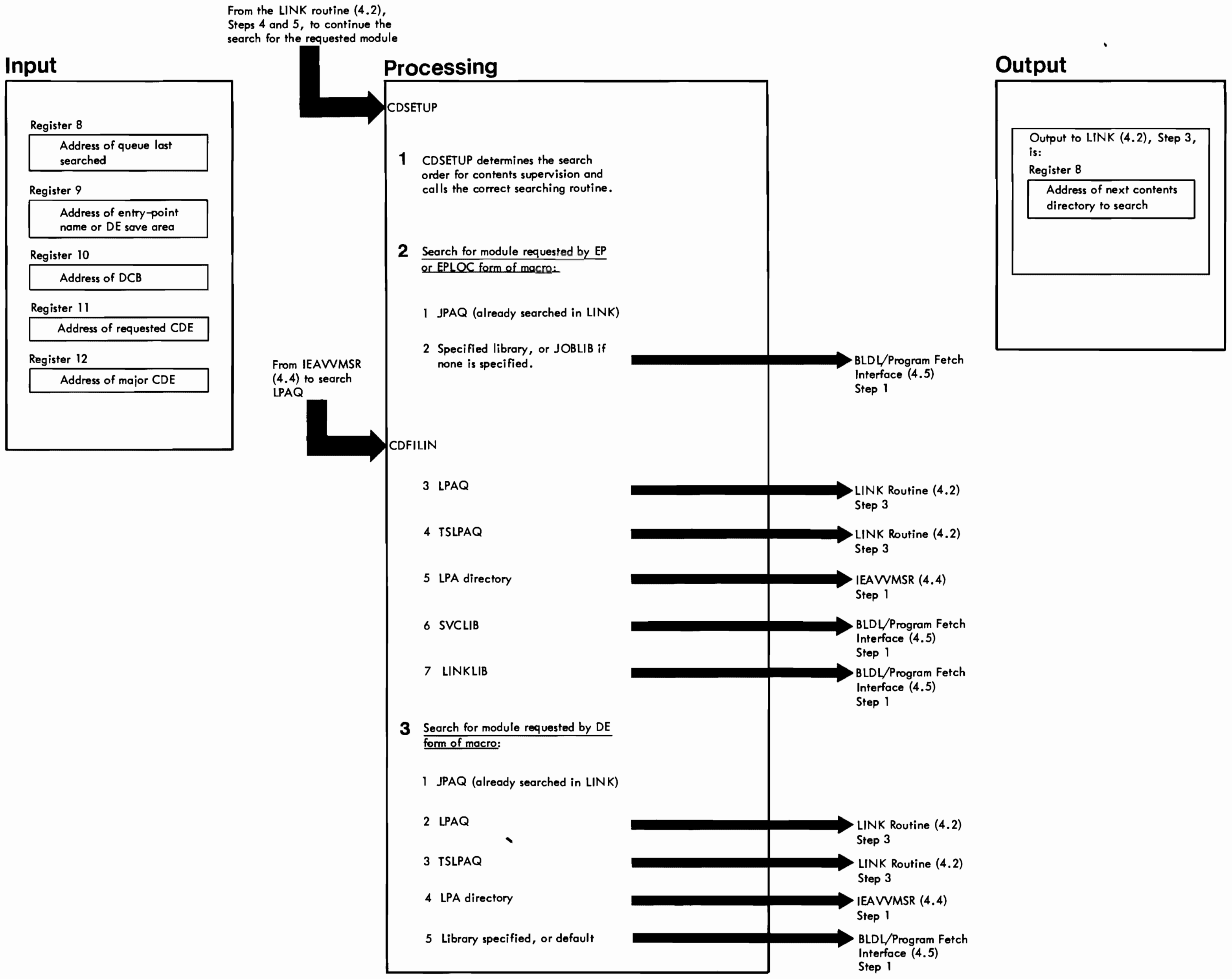
7 If a job-step is being attached, set the JSCB authorized if the CDE is authorized.

Otherwise, exit.

• Diagram 4.2 (Steps 5-7) LINK Routine (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
6 If the CDE (contents directory entry) for the requested module is in the TSLPA (time sharing link pack area), or if the requester is a TSO task, CDMOPUP does not increase the use count.	CDMOPUP	CDMOPUP

Diagram 4.3
Routing to Searching Routines



Input

Register 8
Address of queue last searched

Register 9
Address of entry-point name or DE save area

Register 10
Address of DCB

Register 11
Address of requested CDE

Register 12
Address of major CDE

Processing

From the LINK routine (4.2), Steps 4 and 5, to continue the search for the requested module → CDSETUP

1 CDSETUP determines the search order for contents supervision and calls the correct searching routine.

2 Search for module requested by EP or EPLOC form of macro:

1 JPAQ (already searched in LINK)

2 Specified library, or JOBLIB if none is specified. → BLDL/Program Fetch Interface (4.5) Step 1

From IEAVVMSR (4.4) to search LPAQ → CDFILIN

3 LPAQ → LINK Routine (4.2) Step 3

4 TSLPAQ → LINK Routine (4.2) Step 3

5 LPA directory → IEAVVMSR (4.4) Step 1

6 SVCLIB → BLDL/Program Fetch Interface (4.5) Step 1

7 LINKLIB → BLDL/Program Fetch Interface (4.5) Step 1

3 Search for module requested by DE form of macro:

1 JPAQ (already searched in LINK)

2 LPAQ → LINK Routine (4.2) Step 3

3 TSLPAQ → LINK Routine (4.2) Step 3

4 LPA directory → IEAVVMSR (4.4) Step 1

5 Library specified, or default → BLDL/Program Fetch Interface (4.5) Step 1

Output

Output to LINK (4.2), Step 3, is:

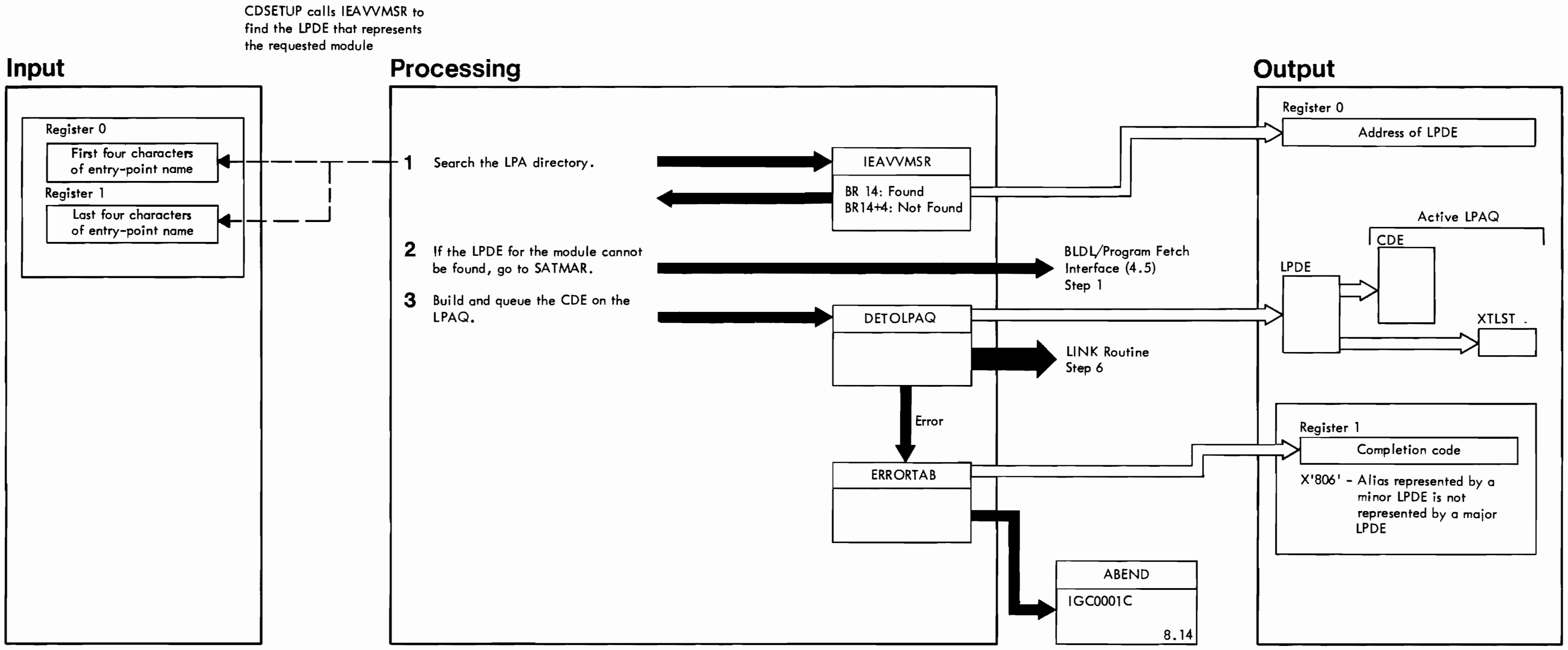
Register 8
Address of next contents directory to search

Diagram 4.3 Routing to Searching Routines (Module IEAVLK01)

NOTES	ROUTINE NAME	LABEL
CDSETUP is a subroutine of contents supervision. It determines the search order for a requested module based on input parameters. The areas searched in Steps 2 and 3 are named in the order that they will be searched based on the type of macro issued.	CDSETUP	CDSETUP

**Diagram 4.4
IEAVVMSR: Searching
the LPA Directory**

CDSETUP calls IEAVVMSR to find the LPDE that represents the requested module



1 Search the LPA directory.

2 If the LPDE for the module cannot be found, go to SATMAR.

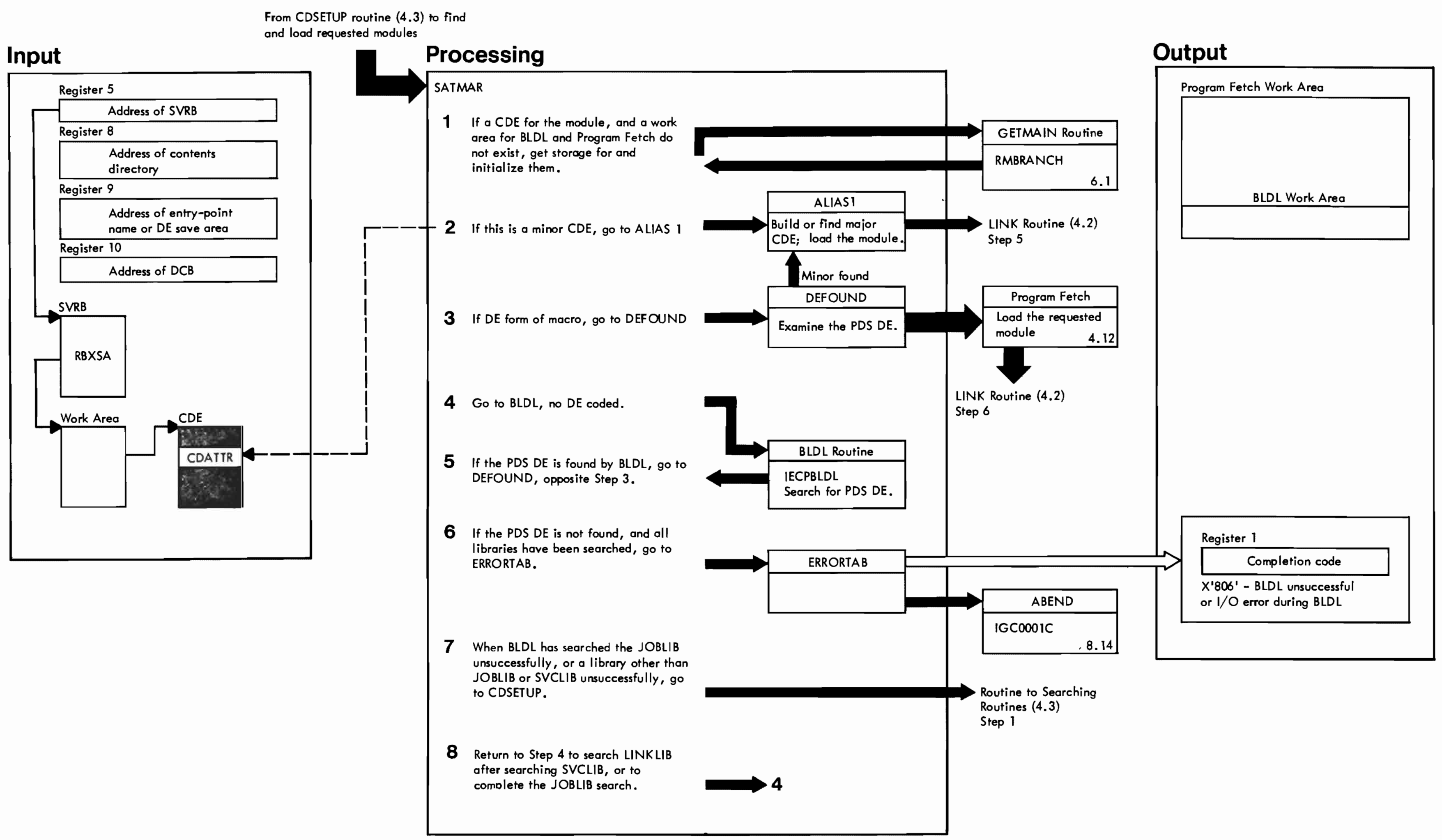
3 Build and queue the CDE on the LPAQ.

X'806' - Alias represented by a minor LPDE is not represented by a major LPDE

Diagram 4.4 IEAVVMSR: Searching the LPA Directory (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
1 IEAVVMSR computes an index factor into the LPA directory by doing an Exclusive OR on the halves of the names in registers 0 and 1, and then dividing the result by the value found in location IEAVVMDI (placed there by NIP). The remainder is multiplied by 32 and added to the address of the LPA directory (in CVTLPDIR). IEAVVMSR now 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 input name.	IEAVVMSR	IEAVVMSR

• Diagram 4.5
BLDL/Program Fetch Interface



1 If a CDE for the module, and a work area for BLDL and Program Fetch do not exist, get storage for and initialize them.

2 If this is a minor CDE, go to ALIAS 1

3 If DE form of macro, go to DEFOUND

4 Go to BLDL, no DE coded.

5 If the PDS DE is found by BLDL, go to DEFOUND, opposite Step 3.

6 If the PDS DE is not found, and all libraries have been searched, go to ERRORTAB.

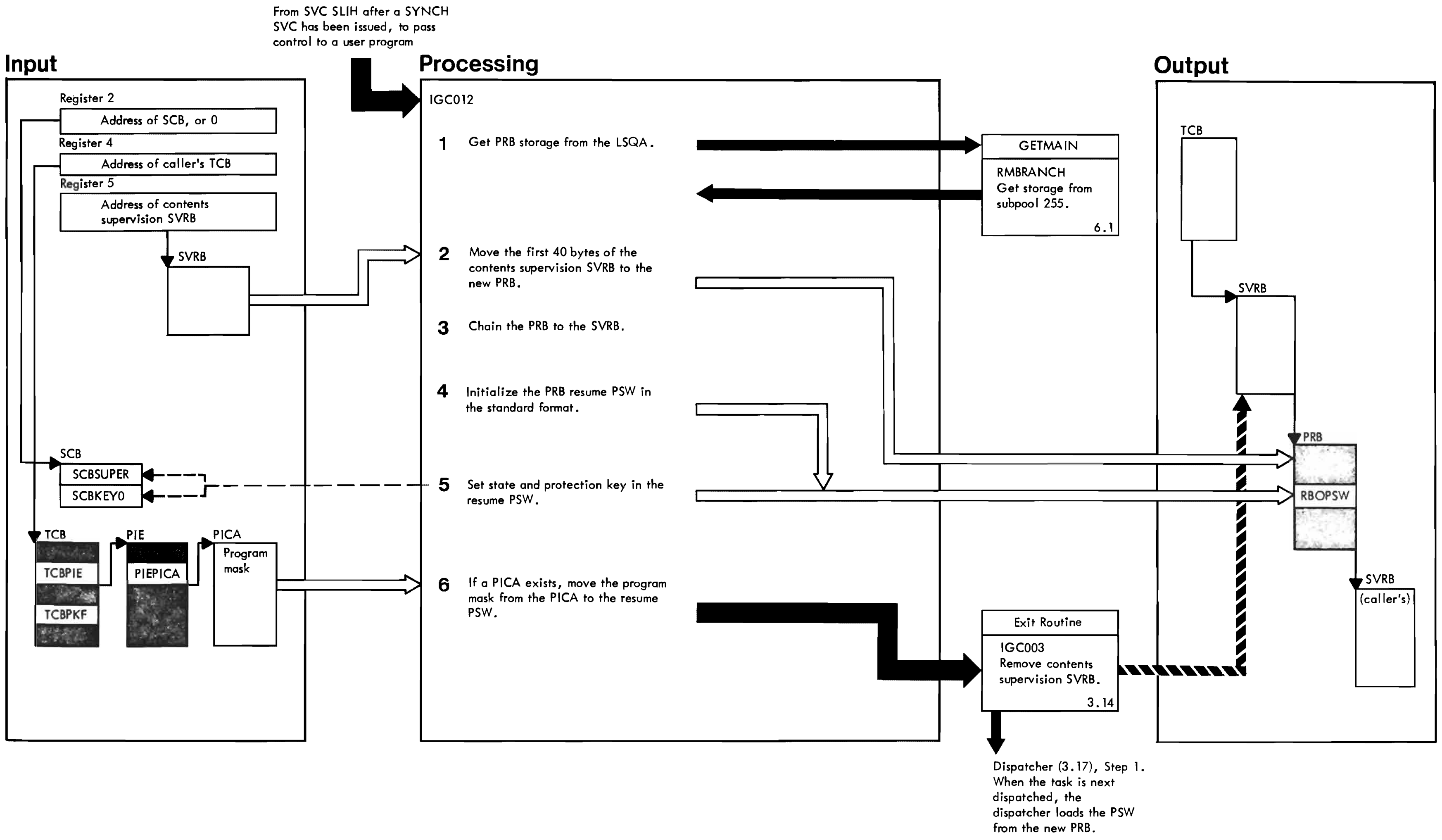
7 When BLDL has searched the JOBLIB unsuccessfully, or a library other than JOBLIB or SVCLIB unsuccessfully, go to CDSETUP.

8 Return to Step 4 to search LINKLIB after searching SVCLIB, or to complete the JOBLIB search.

Diagram 4.5 BIDL/Program Fetch Interface (Module IEAVLK01)

NOTES	ROUTINE NAME	LABEL
1 SATMAR creates a CDE (contents directory entry) and queues it to the job-step TCB. The CDE is built prior to BLDL and Program Fetch processing to ensure that subsequent requests for the same module will be deferred during BLDL or Program Fetch processing.	SATMAR	SATMAR
4 BUILDEL is a contents supervision subroutine that calls the BLDL routine. BLDL trys to find the PDS DE (partitioned data set directory entry) for the requested module.	BUILDEL	BUILDEL

Diagram 4.6 SYNCH Routine

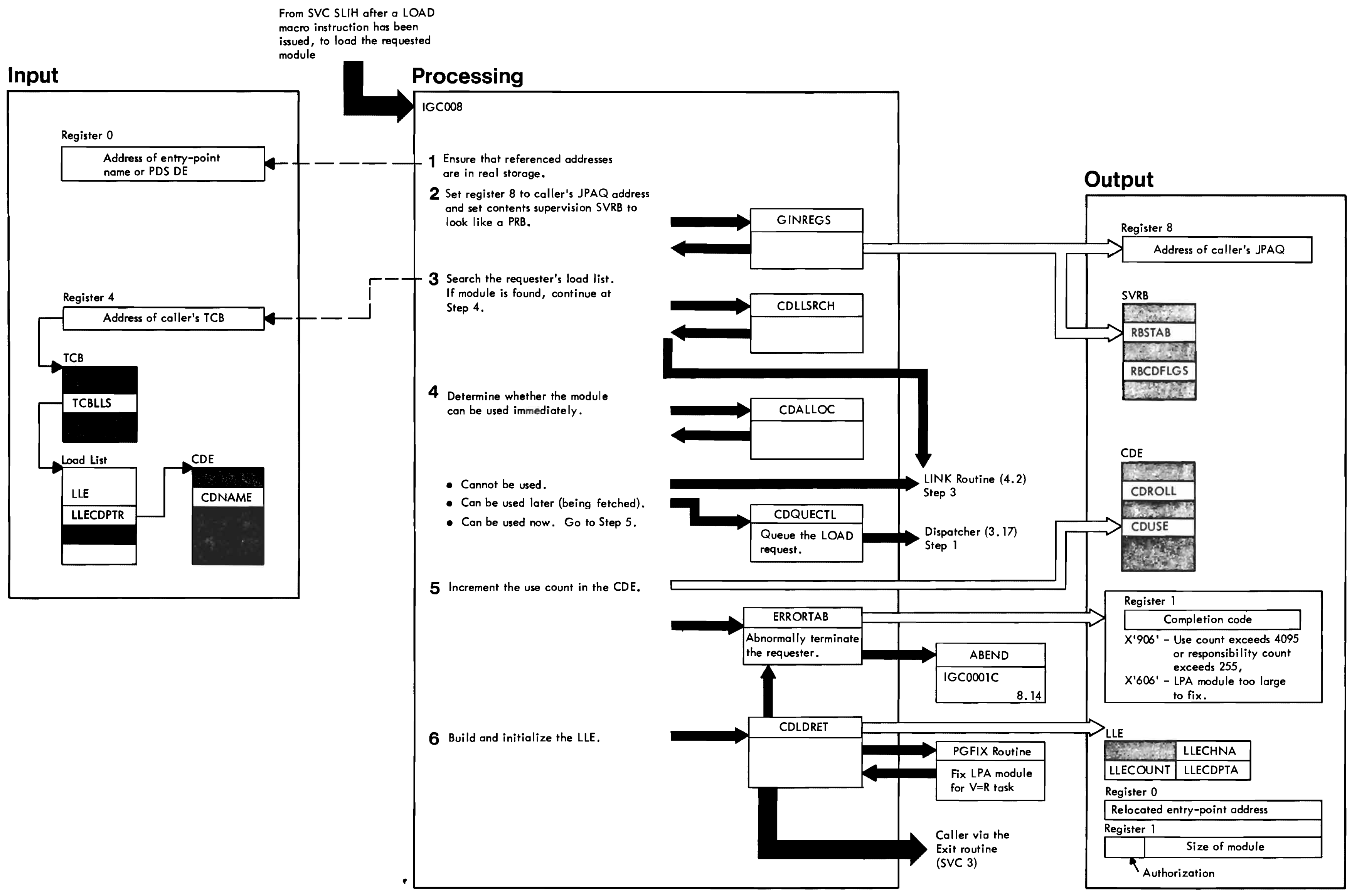

From SVC SLIH after a SYNCH SVC has been issued, to pass control to a user program
Input
Register 2
Address of SCB, or 0
Register 4
Address of caller's TCB
Register 5
Address of contents supervision SVRB
SVRB
SCB
SCBSUPER
SCBKEY0
TCB
TCBPIE
TCBPKF
PIE
PIEPICA
PICA
Program mask
Processing
IGC012
GETMAIN
RMBRANCH
Get storage from subpool 255.
6.1
Exit Routine
IGC003
Remove contents supervision SVRB.
3.14
Dispatcher (3.17), Step 1. When the task is next dispatched, the dispatcher loads the PSW from the new PRB.
Output
TCB
SVRB
PRB
RBOPSW
SVRB
(caller's)

1. Get PRB storage from the LSQA.
2. Move the first 40 bytes of the contents supervision 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.

Diagram 4.6 SYNCH Routine (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
The SYNCH routine makes it possible for a supervisor routine to take a synchronous exit to a user program. SYNCH initializes a PRB and schedules execution of the requested program. After the user program has been executed, the supervisor routine that issued the SYNCH macro instruction regains control.	SYNCH	IGC012
1 After minor housekeeping processing, SYNCH passes CDEPILOG THRUX control to CDEPILOG at THRUX.		
2 The PRB is chained behind the SVRB because the Exit routine later removes the top RB on the RB queue and the module represented by the request level RB gains control.		
4 Standard format for the PSW is X'070D000000'. This may be modified in Steps 5 and 6. The RBOPSW is called the resume PSW in the steps above. The dispatcher loads this PSW to give control to the user program.		
5 If the SYNCH request was not to enter a STAE exit routine (TCBSYNCH=0), the requested program will execute with the protection key in the caller's TCB (TCBPKF) and in problem state.		
If the SYNCH request was to enter a STAE exit routine, the RBOPSW is set to indicatee supervisor state if SCBSUPER equals 1, and key 0 if SCBKEY0 equals 1. If SCBSUPER equals 0, the RBOPSW is set tc indicate problem state, and if SCBKEY0 equals 0, the RBOPSW is set equal to the protection key in the caller's TCB.		

Diagram 4.7
LOAD Routine



1 Ensure that referenced addresses are in real storage.

2 Set register 8 to caller's JPAQ address and set contents supervision SVRB to look like a PRB.

3 Search the requester's load list. If module is found, continue at Step 4.

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 Increment the use count in the CDE.

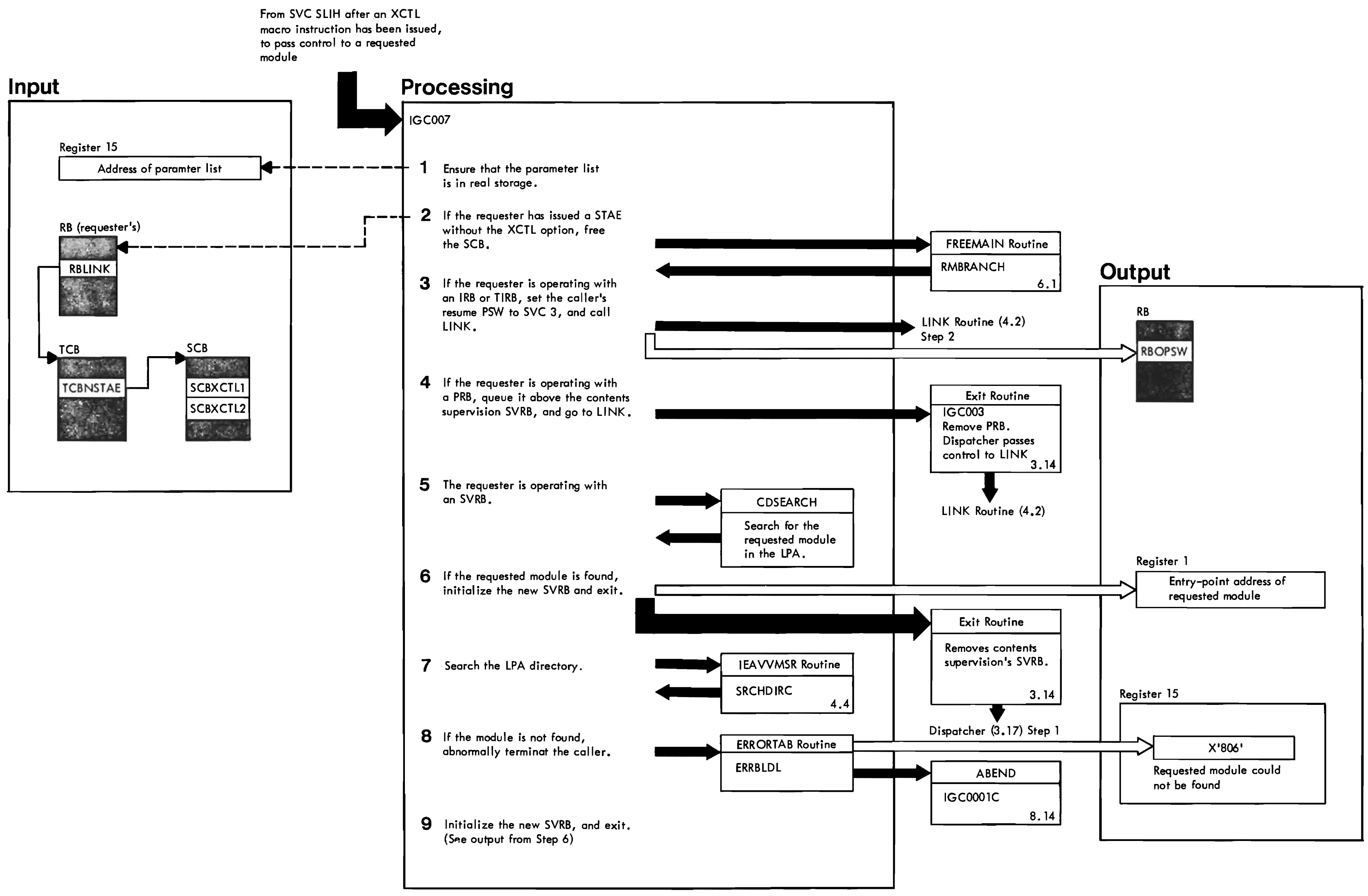
6 Build and initialize the LLE.

X'906' - Use count exceeds 4095 or responsibility count exceeds 255,
X'606' - LPA module too large to fix.

• Diagram 4.7 LOAD Routine (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
The LOAD routine brings a module containing a specified entry point into virtual storage if a usable copy is not available. It increases by 1 the responsibility and use counts, and returns control to the requester.	LOAD	IGC008
1 LOAD calls DALPRFIX to ensure that the address of the entry-point name or PDS DE for the requested module is in real storage.	DALPRFIX	DALPRFIX
2 The two high order bytes of the RBSTAB field in the SVRB (supervisor request block) are set to 1 to make the SVRB look like a PRB (program request block). Now ABEND can free the SVRB if the requester abnormally terminates. The address of the requesting JPAQ (job pack area queue) is placed in register 8 in case LINK must continue to search for the module. Also, LOAD sets the lower order bit in RBCDFLGS equal to 1 to indicate that this is a load request.	GINREGS	GINREGS
5 If the CDE for the requested module is in the TSLPA and if the requester is a TSO task, CDMOPUP does not increase the use count in the CDE. If the use count exceeds 4095, the requester is abnormally terminated.	CDMOPUP	CDMOPUP
6 If no LLE exists, CDLDRET gets storage for an LLE and chains it to the caller's load list. If the caller is a nonpageable task and the requested module is in the LPA, the requested module is fixed by calling the PGFIX routine. CDLDRET increases the responsibility count in the LLE, and if it exceeds 255, the requester is abnormally terminated.	CDLDRET	CDLDRET

Diagram 4.8 XCTL Routine



From SVC SLIH after an XCTL macro instruction has been issued, to pass control to a requested module

IGC007

1 Ensure that the parameter list is in real storage.

2 If the requester has issued a STAE without the XCTL option, free the SCB.

3 If the requester is operating with an IRB or TIRB, set the caller's resume PSW to SVC 3, and call LINK.

4 If the requester is operating with a PRB, queue it above the contents supervision SVRB, and go to LINK.

5 The requester is operating with an SVRB.

6 If the requested module is found, initialize the new SVRB and exit.

7 Search the LPA directory.

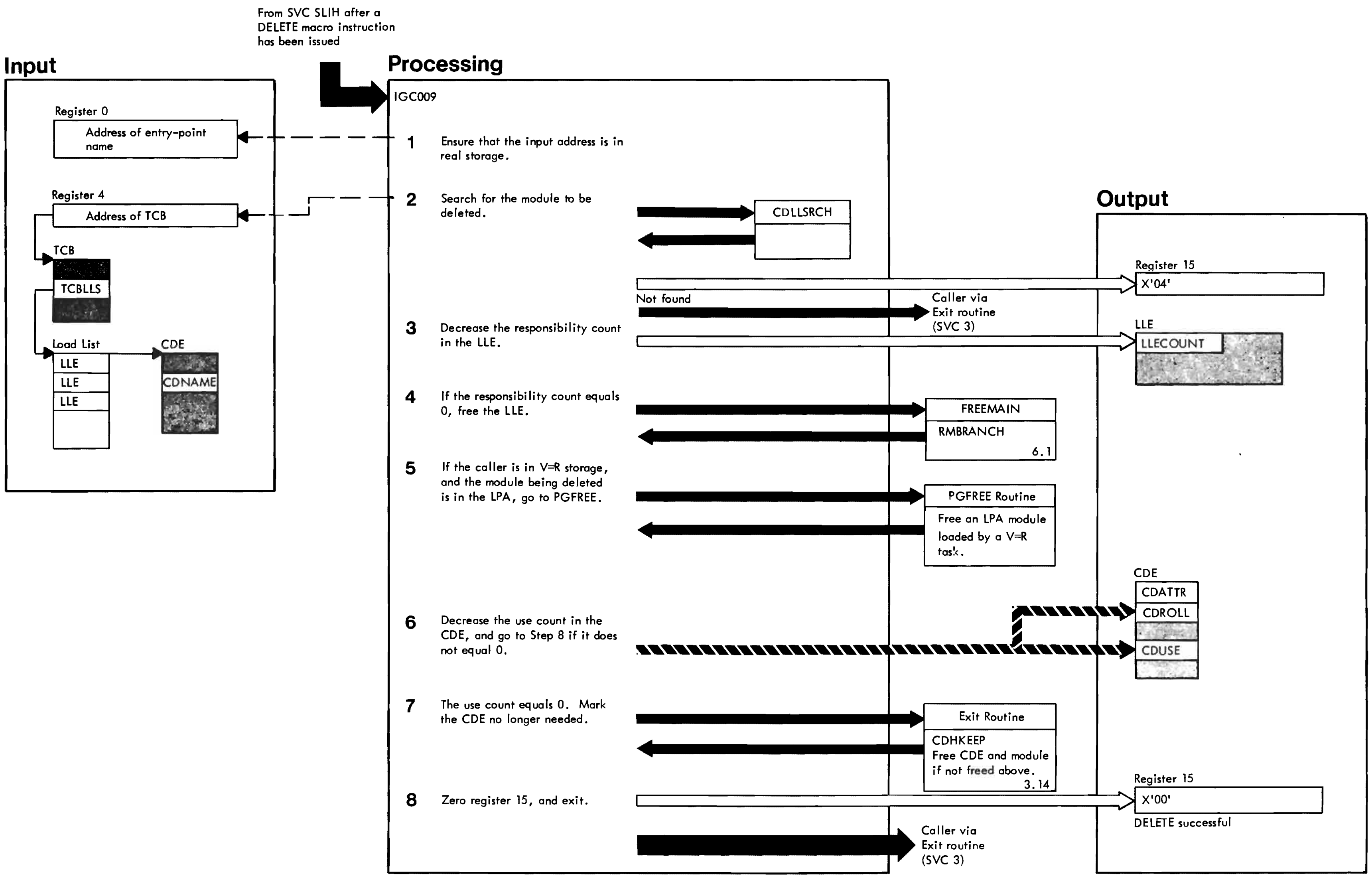
8 If the module is not found, abnormally terminat the caller.

9 Initialize the new SVRB, and exit. (See output from Step 6)

Diagram 4.8 XCTL Routine (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
The XCTL routine provides linkage to a specified load module. The requested module is executed with the same key and in the same state as the requester. The XCTL routine itself completely services only XCTL requests for callers represented by an SVRB. XCTL calls the LINK routine to honor requests made by callers operating with a PRB or IRB.	XCTL	IGC007
XCTL ensures that the requester does not regain control after the requested module has been executed.		
1 XCTL calls LXPREFIX to ensure that the parameter list is in real storage.	LXPREFIX	LXPREFIX
2 If the SCBXCTL2 flag in the SCBSTA is equal to 1, the caller issued a STA with XCTL option before issuing an XCTL. XCTL frees STAE SCBs that do not indicate the XCTL option.	XCTL	STAELOOP
XCTL marks as noncancellable (sets SCBXCTL1=1) all STAESCBs that indicate an XCTL option, and were not issued by a supervisor program. The routine does this to prevent the Exit routine from freeing them.		
3 Setting the resume PSW to an SVC 3 instruction causes the requester to exit when the requested module exits. A requester represented by a TIRB follows this path to the CDADVANS entry point in LINK.		IRBPROC
4 XCTL moves the caller's PRB ahead of the contents supervision SVRB on the RB queue. The Exit routine removes the PRB, and when the task is next dispatched, processing continues at CDADVANS.		PRBPROC
6 The XCTL routine sets the resume PSW (RBOPSW) in the requested module's SVRB to the module's entry-point address.		FOUNDUM
9 Same as Step 6.		FOUNDEM

Diagram 4.9
DELETE Routine



1 Ensure that the input address is in real storage.

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 If the caller is in V=R storage, and the module being deleted is in the LPA, go to PGFREE.

6 Decrease the use count in the CDE, and go to Step 8 if it does not equal 0.

7 The use count equals 0. Mark the CDE no longer needed.

8 Zero register 15, and exit.

Diagram 4.9 DELETE Routine (Module IEAVLK00)

NOTES	ROUTINE NAME	LABEL
When a module brought into virtual storage via the LOAD routine is no longer needed by the program that requested it, the requesting program issues a DELETE macro instruction. The DELETE routine then decreases by one the responsibility count in the load list element (LLE). When the responsibility count reaches zero, DELETE frees the LLE. If the module being deleted is not a TSO module or represented by a CDE in the TSLPA, the DELETE routine also decreases the use count in the contents directory entry (CDE).	DELETE	IGC009
1 DELETE calls DALPRFIX to ensure that the address of the requested module's entry-point name is in real storage.	DALPRFIX	DALPRFIX
3 The LLE responsibility count is a record of the number of outstanding LOAD requests for the module.		
6 The CDE use count represents the total number of requests for a program via ATTACH, LINK, LOAD, and XCTL macro instructions. The count increases each time one of these macros is successfully issued, and decreases each time a DELETE is issued.		
The CDUSE and CDROLL fields contain the use count. The overflow from CDUSE is in CDROLL. When the count in these fields is zero, DELETE sets the CDATTR field to indicate that the module is no longer usable. In this case, the CDHKEEP routine frees the real storage occupied by the program, its extent list, and its major and minor CDEs.		

Diagram 4.10
IDENTIFY Routine

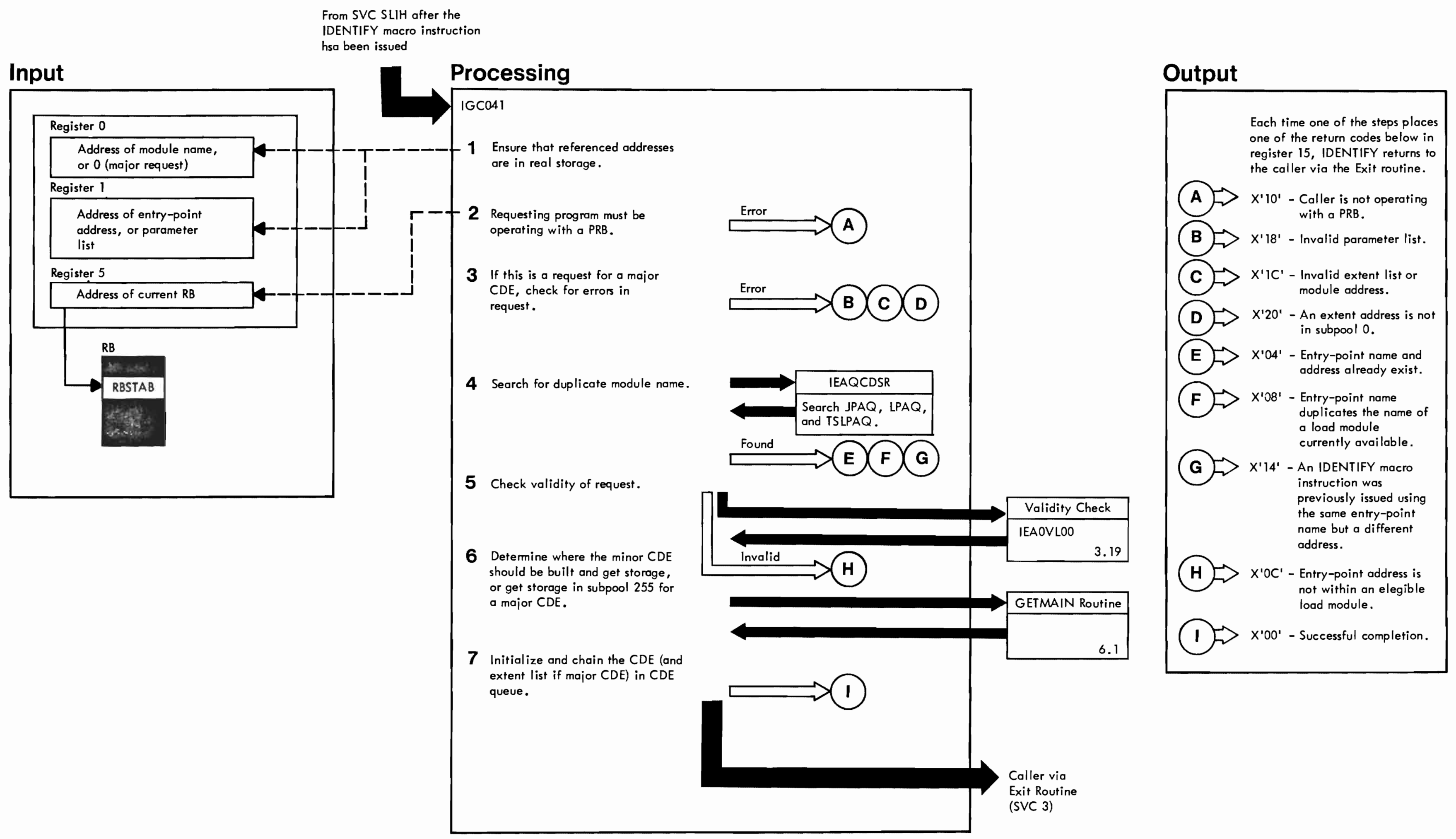
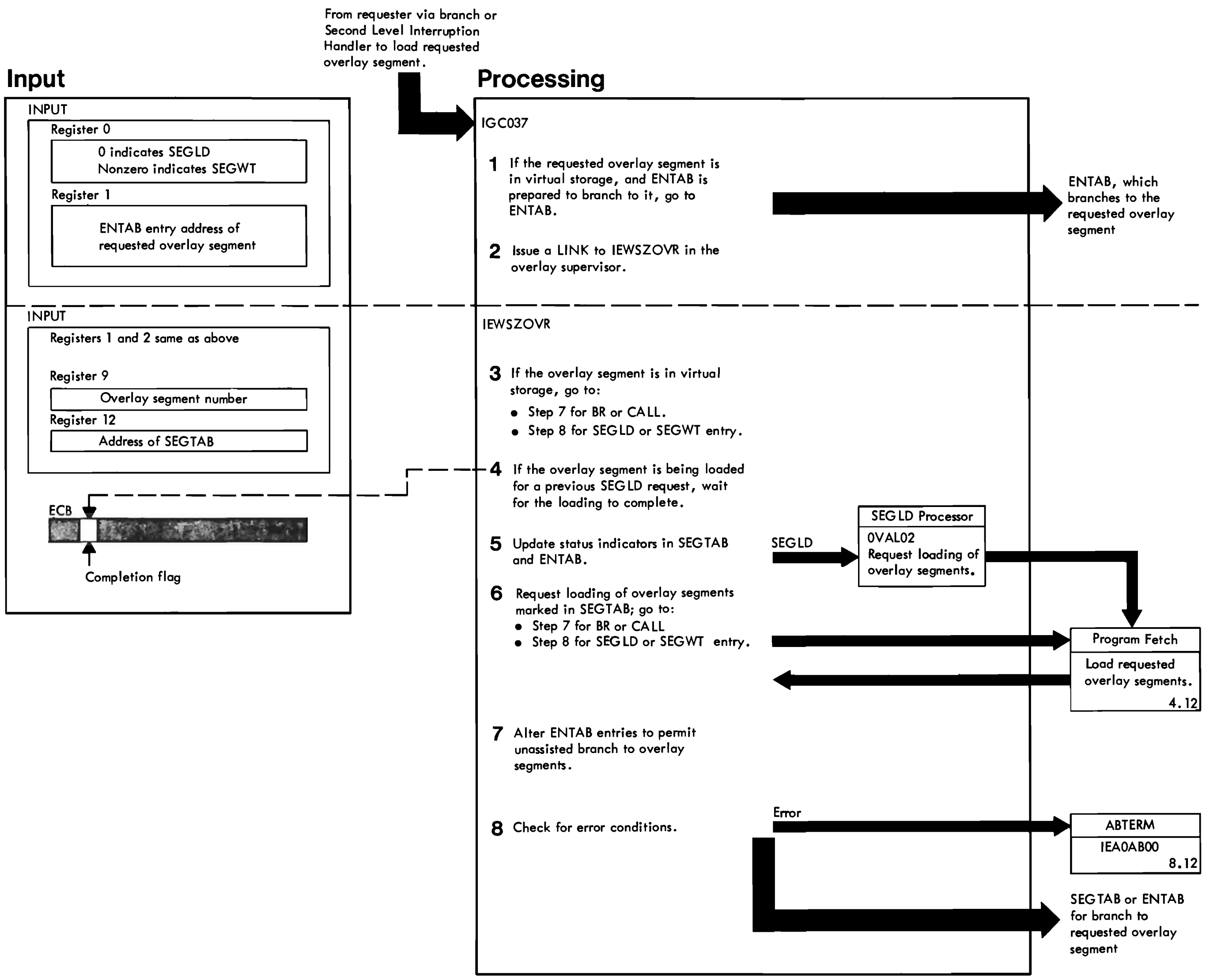

From SVC SLIH after the IDENTIFY macro instruction hsa been issued
Input
Register 0
Address of module name, or 0 (major request)
Register 1
Address of entry-point address, or parameter list
Register 5
Address of current RB
RB
RBSTAB
Processing
IGC041
1 Ensure that referenced addresses are in real storage.
2 Requesting program must be operating with a PRB.
Error
A
3 If this is a request for a major CDE, check for errors in request.
Error
B
C
D
4 Search for duplicate module name.
IEAQCDSR
Search JPAQ, LPAQ, and TSLPAQ.
Found
E
F
G
5 Check validity of request.
Validity Check
IEAOVL00
3.19
6 Determine where the minor CDE should be built and get storage, or get storage in subpool 255 for a major CDE.
Invalid
H
GETMAIN Routine
6.1
7 Initialize and chain the CDE (and extent list if major CDE) in CDE queue.
I
Caller via Exit Routine (SVC 3)
Output
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'20' - An extent address is not in subpool 0.
E X'04' - Entry-point name and address already exist.
F X'08' - Entry-point name duplicates the name of a load module currently available.
G X'14' - An IDENTIFY macro instruction was previously issued using the same entry-point name but a different address.
H X'0C' - Entry-point address is not within an elegible load module.
I X'00' - Successful completion.

Diagram 4.10 IDENTIFY Routine (Module IEAVID00)

NOTES	ROUTINE NAME	LABEL
The IDENTIFY routine informs the supervisor of a module's embedded entry-point name (a name that was not established by the linkage editor). IDENTIFY informs the supervisor by creating a minor CDE (contents directory entry) to represent the embedded entry-point name.	IDENTIFY	IGC041
The IDENTIFY routine can also build a major CDE and an extent list in subpool 255 for a module brought directly into storage by the loader. This allows the supervisor to identify the module.		
1 If register 0 equals 0, a major CDE is requested. In this case, IDPREFIX ensures that the parameter list or entry-point address in register 1 is in real storage.	IDPREFIX	IDPREFIX
If register 0 contains an address, IDPREFIX ensures that the address of the module name is in real storage.		
3 Beginning with this step, all processing to build a major CDE is handled by a subroutine of IDENTIFY, called MAJORCDE. Because MAJORCDE performs the same functions as IDENTIFY does to build a minor CDE, it is shown in Steps 4-7 above along with IDENTIFY processing.	MAJORCDE	MAJORCDE
6 Minor CDEs may be built in the JPAQ, TSLPAQ, or LPAQ, depending upon where the major CDE is located.		



From requester via branch or Second Level Interruption Handler 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

Processing

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.

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.

ENTAB, which branches to the requested overlay segment
SEGLD
SEGLD Processor
0VAL02
Request loading of overlay segments.
Program Fetch
Load requested overlay segments.
4.12
Error
ABTERM
IEA0AB00
8.12
SEGTAB or ENTAB for branch to requested overlay segment

Diagram 4.11 Overlay Supervisor (Modules IEWSUOVR and IEWSWOVR)

NOTES	ROUTINE NAME	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. 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.	Overlay Supervisor	IGC037

NOTES	ROUTINE NAME	LABEL
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).	Overlay Supervisor	

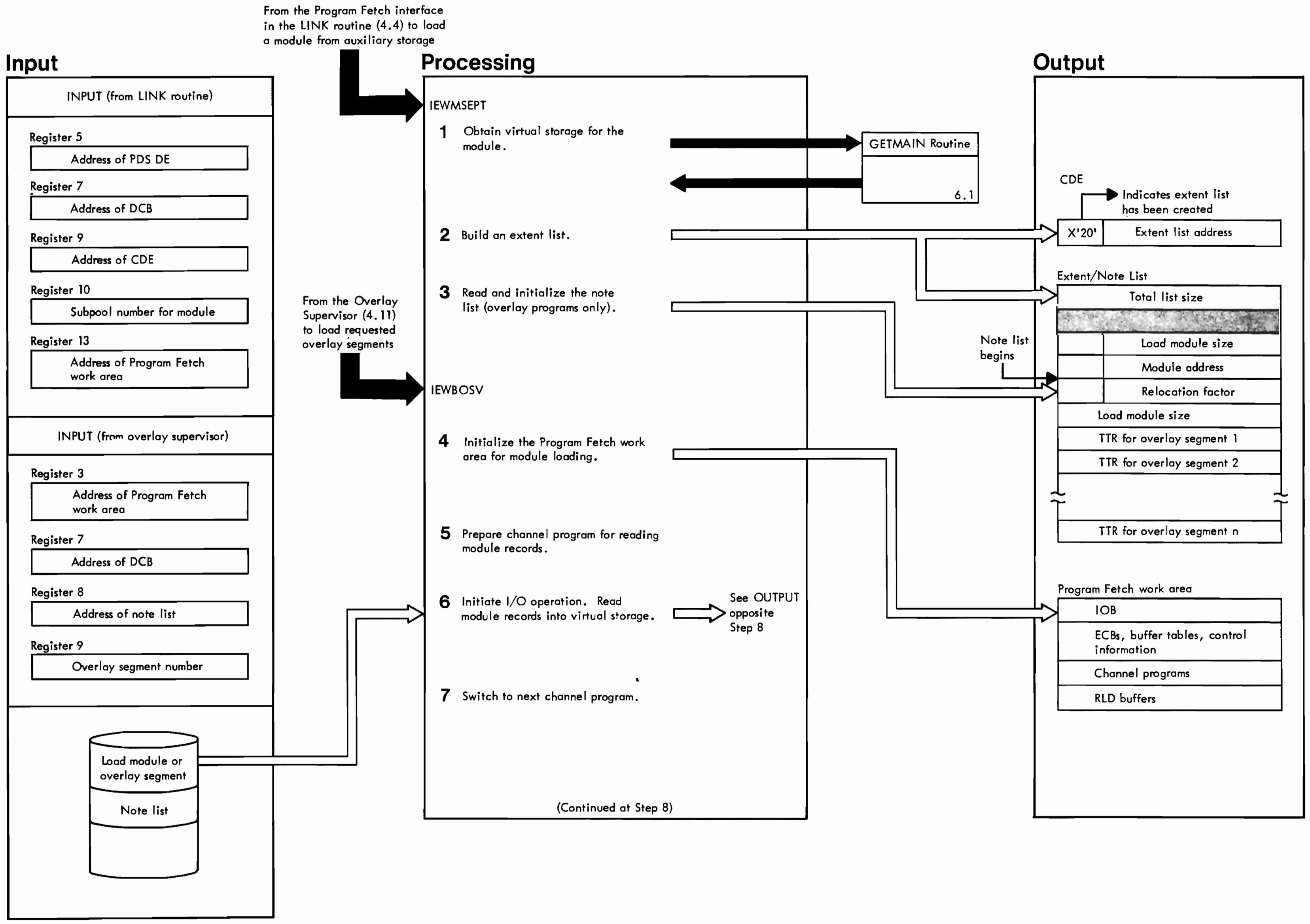

From the Program Fetch interface in the LINK routine (4.4) to load a module from auxiliary storage
Input
INPUT (from LINK routine)
Register 5
Address of PDS DE
Register 7
Address of DCB
Register 9
Address of CDE
Register 10
Subpool number for module
Register 13
Address of Program Fetch work area
INPUT (from overlay supervisor)
Register 3
Address of Program Fetch work area
Register 7
Address of DCB
Register 8
Address of note list
Register 9
Overlay segment number
Load module or overlay segment
Note list
From the Overlay Supervisor (4.11) to load requested overlay segments
Processing
IEWMSEPT
1 Obtain virtual storage for the module.
2 Build an extent list.
3 Read and initialize the note list (overlay programs only).
IEWBOSV
4 Initialize the Program Fetch work area for module loading.
5 Prepare channel program for reading module records.
6 Initiate I/O operation. Read module records into virtual storage.
See OUTPUT opposite Step 8
7 Switch to next channel program.
(Continued at Step 8)
GETMAIN Routine
6.1
Output
CDE
Indicates extent list has been created
X'20'
Extent list address
Extent/Note List
Total list size
Note list begins
Load module size
Module address
Relocation factor
Load module size
TTR for overlay segment 1
TTR for overlay segment 2
TTR for overlay segment n
Program Fetch work area
IOB
ECBs, buffer tables, control information
Channel programs
RLD buffers

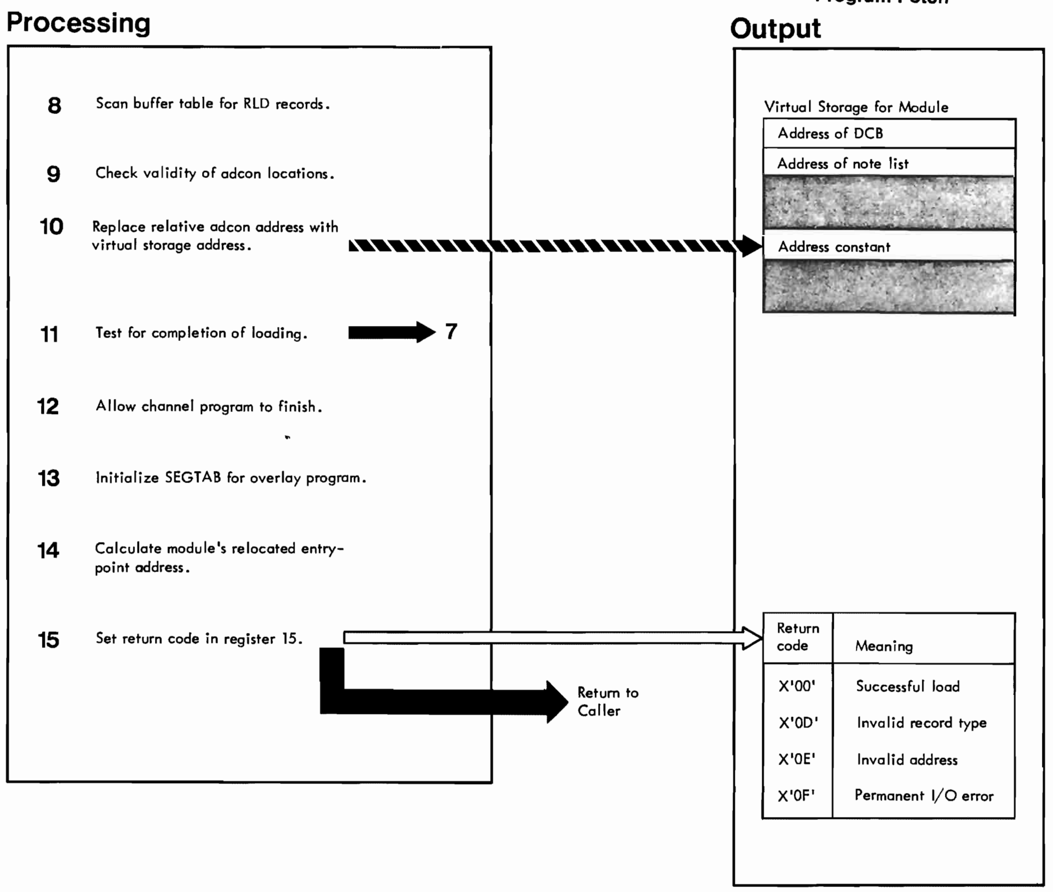
Diagram 4.12 (Steps 1-7) Program Fetch (Module IEWFETCH)

NOTES	ROUTINE NAME	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. The subroutines of contents supervisor that search for requested modules and the overlay supervisor use Program Fetch to load modules. The searching subroutines of contents supervision 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 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 auxiliary storage addresses to 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.	Program Fetch	IEWMSEPT
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 places it in the CDE.		

NOTES	ROUTINE NAME	LABEL
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 EXCPVR channel request to read the note list into virtual storage. After the note list has been read, Program Fetch builds a note list prefix that it uses when called to load an overlay segment.	Program Fetch	
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 is needed by the I/O supervisor. • Two event control blocks (ECBs). One ECB is posted by the I/O supervisor when a channel-end condition occurs. The other is posted by a PCI appendage routine when a program-controlled interruption occurs in a channel program. The posting of either ECB permits the Program Fetch to be restarted after an I/O wait interval. • 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. In addition, Program Fetch requests storage for another work area if the DCB (data control block) does not reference either SYS1.LINKLIB, SYS1.SVCLIB, or JOBLIB.		

NOTES	ROUTINE NAME	LABEL
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 <u>Preparing for Execution of a Channel Program</u> 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), 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 is 18K bytes long, unless the end of the module is encountered first.		
6 Program Fetch starts a channel program by issuing an EXCPVR macro instruction to obtain supervisor linkage to the I/O supervisor. The IOB address is provided as an operand of the macro instruction.		
The I/O supervisor then passes control to the page fix appendage (PGFX). The PGFX appendage creates a fix list in the fetch work area. This list contains doubleword entries; each entry gives the low and high virtual addresses of storage to be fixed for the I/O request. In this manner, page faults are avoided when the I/O supervisor or appendages address the fixed storage. Included in the list are:		
• DCB for the module's data set.		
• The text buffer defined before issuing EXCPVR.		

NOTES	ROUTINE NAME	LABEL
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. Once the fix list is created, the I/O supervisor regains control; register 10 contains the address of the fix list, and register 11 contains the number of entries in the list.	Program Fetch	
Before starting the channel program, the I/O supervisor enters the Fetch Start I/O Appendage routine. On the first entry to the Start I/O Appendage for a loading operation, all channel programs are translated. On subsequent entries for the same load operation, the only CCWs requiring translation are those that read the module text records. The other CCWs all address the nonpageable fetch work area, so they are valid unless the page containing the work area has been swapped into a new real page. When a new page is encountered, the Start I/O Appendage translates all channel programs again.		
The text CCWs are treated separately and are translated on each entry to the appendage. 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. Control is then returned to the I/O supervisor.		
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 IOB. 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 IOB. 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 I/O supervisor or an appendage routine. Such posting indicates that one or two records have been read and that further processing can occur in Program Fetch.		



Processing

8 Scan buffer table for RLD records.

9 Check validity of adcon locations.

10 Replace relative adcon address with virtual storage address.

11 Test for completion of loading. → 7

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 to Caller

Output

Return code	Meaning
X'00'	Successful load
X'0D'	Invalid record type
X'0E'	Invalid address
X'0F'	Permanent I/O error

Diagram 4.12 (Steps 8-15) Program Fetch (Module IEWFETCH)

NOTES	ROUTINE NAME	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 PCI appendage routine. The appendage examines the current RLD buffer to determine the channel program switching required, and operates as follows: • If the next RLD buffer is filled, the next channel program cannot be used, so chaining is suppressed and a "buffers full" condition is indicated. The current channel program is not altered. • 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. In all the above cases, the fetch ECB is posted to prepare the Program Fetch routine for restart by the dispatcher, and control is returned to the I/O supervisor.	Program Fetch	
The Channel-End Appendage routine is entered by the I/O supervisor when the channel program has terminated. The appendage returns control to the I/O supervisor to post the I/O ECB when channel end is due to the fact that: • The entire module or segment has been loaded. • A buffer-full condition is indicated. • A permanent I/O error has occurred. When none of the above conditions is present, channel end occurred because the TIC instruction was stored by the PCI Appendage after the channel had fetched the NOP CCW. In this case, the Channel-End Appendage routine returns control to the I/O supervisor to restart the channel program.	Program Fetch	
11 Program Fetch is restarted after the PCI Appendage routine or the I/O supervisor has posted an ECB. 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. The subroutine searches for a buffer table entry whose "busy" indicator is set. The indication means that the associated buffer contains an RLD record. When such a buffer is found, the relocation subroutine relocates each address constant specified in the record. (When RLD records in all "busy" buffers have been processed, Program Fetch either restarts a channel program if the "buffer full" flag is on, or issues a WAIT macro instruction to await the loading of another 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 PCI Appendage 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 its processing or waits, depending on whether all I/O operations have been completed. When all I/O operations have been completed, the subroutine 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.		

SECTION 5

Paging Supervision

5

The VS2 system enables programs to occupy an address range greater than the address range of real storage. This address space is called virtual storage and is divided into 4K blocks called pages. These pages may reside in a portion of auxiliary storage called external page storage, in page frames in real storage, in neither, or in both. The paging supervisor manages the exchange of virtual pages between real and external page storage.

Whenever a program references a virtual address contained in a page that is not in real storage, the CPU generates an interruption which causes control to be given to the paging supervisor. The paging supervisor moves a copy of the needed page into a 4K page frame in real storage. When the page is in real storage, the referenced virtual address is translated into the real storage address of the data. The entire paging operation and translation process are transparent to the programmer, allowing him to use the virtual address space as though it were real storage.

ORGANIZATION OF THE PAGING SUPERVISOR

The paging supervisor is composed of four subcomponents:

- Real Storage Administration, which maintains a record of the status of all pages in real storage and processes any request involving the allocation, release, or change in status of a real storage page.
- Page Administration, which processes all requests for movement of pages from real storage to external page storage and vice-versa.
- Interface Control, which controls the flow of a request through the paging supervisor, provides interfaces to the paging supervisor service routines, and controls most of the external interfaces into the paging supervisor.
- Auxiliary Storage Administration, which manages the external page portion of auxiliary storage.

See "Section 10: Program Organization" for a list of modules in each subcomponent and the routines within each module.

ENTRIES TO THE PAGING SUPERVISOR

The paging supervisor receives control in one of five ways:

1. When a missing page interruption occurs, the paging supervisor receives control through the Program First-Level Interruption Handler. (A missing page interruption, also called a page fault, indicates that a page containing a referenced virtual address is not in real storage.)
2. When pages must be moved between external page storage and real storage, the paging supervisor receives control from the dispatcher through normal dispatching procedures. (A special task, called the paging task, runs as the highest priority task in the system, and handles delayed [queued] paging requests. Because the paging task is the highest priority task, the system is able to save CPU time by bypassing the Task Switch routine when the paging task must be dispatched.)
3. Upon occurrence of a channel end interruption (CE) or an I/O error on a previously started paging operation, the paging supervisor receives control through an I/O supervisor appendage.
4. When an SVC instruction requesting a paging supervisor service is executed, the paging supervisor receives control from the SVC FLIH.
5. The paging supervisor also receives control when a branch entry is made into a paging supervisor service routine.

See Diagrams 5.1 through 5.5 for the functional flow through the paging supervisor after each type of entry.

PAGING SUPERVISOR ALGORITHMS

The paging supervisor attempts to make the most efficient use of real storage by keeping only needed pages in real storage and by moving pages no longer needed to auxiliary storage devices. It also tries to conserve CPU time by paging only when necessary and by limiting the amount of time that can be spent on paging operations. To do these things, the paging supervisor uses two algorithms: the page replacement algorithm and the task disablement algorithm.

The Page Replacement Algorithm

To facilitate moving pages into real storage, the paging supervisor maintains a list of page frames that are not being used. This list is called the available page queue.

Whenever the number of available page frames (the available page count, or APC) falls below a predetermined value, the paging supervisor replenishes the available page queue by stealing a page frame from a program. The PFTE (page frame table entry) representing the stolen page is moved from one of the queues representing page frames in use to the available page queue. If the page occupying the page frame has been changed, the page is moved to external page storage, replacing the previous external storage copy before the page frame is made available. The stealing of pages continues until the APC has been incremented to the predetermined value.

The paging supervisor uses the page replacement algorithm to decide which page frames are to be stolen. This algorithm reduces the number of page-outs by attempting to select the least-used pages and by minimizing the stealing of changed pages. See Figure 5-1 for an example of the operation of the page replacement algorithm.

Page frames are selected for reallocation according to the settings of the reference and change bits. These bits are part of the storage keys for the page frame and are modified by the CPU each time a page is referenced or changed. By examining these bits, the paging supervisor determines which pages are least needed (assuming the page that has not been referenced for the longest time is the least needed), and makes the page frame it occupies available for reallocation. The algorithm is biased to select unchanged pages whenever possible because the page frames that they occupy can be made available immediately: there is no need to wait for the page to be written to external page storage.

Six queues of PFTEs (which represent page frames) are used: an available page queue, a hold page queue, and four active page queues. The available page queue contains the PFTEs for the page frames that are available for immedate allocation. The hold page queue contains PFTEs for page frames that were recently allocated and are therefore unlikely candidates for replacement. The four active page queues represent all page frames that are now in use by programs but can be made available for reallocation. (PFTEs can also reside on a seventh queue, the SQA/LSQA reserve queue, which is not used for page replacement.)

There is one active page queue for each possible configuration of the reference and change bits. Reference and change bit settings are denoted as R,C so that 0,0 indicates that the virtual page has neither been referenced nor changed. The meanings of the four possible bit configurations are:

R,C	Meaning
0,0	The virtual page has not been referenced since the last inspection, and has not been changed.
0,1	The virtual page has not been referenced since the last inspection, but was changed at some previous time.
1,0	The virtual page has been referenced since the last inspection, but has not been changed.
1,1	The virtual page has been referenced since the last inspection, and has been changed.

Each queue is maintained in a first-in, first-out order, thus preserving a record of the comparative longevity for each page.

The reference and change bits are set automatically by the CPU. Since pages are constantly being referenced and changed, the bit settings may not be the same when interrogated as when the PFTE was first placed on a particular queue. As the paging supervisor serially searches the queues, it steals those page frames whose status has remained unchanged. Those page frames whose status has changed (that is, their reference and change bits are not the same as those of the active page queue they are presently on) are moved to the appropriate queue. However, before placing the PFTE on its new queue, the paging supervisor resets the reference bit to zero. It is this resetting of the reference bit to zero that makes unreferenced but changed pages (0,1) possible. The change bit is never reset during PFTE movement.

The processing of this algorithm is performed by the Page Replacement routine (Diagram 5.15).

The Task Disablement Algorithm

The number of paging operations, and consequently the amount of time spent paging, depend on the number of programs active in the system and the number of paging requests they generate. If too much time is spent paging, system performance can be seriously degraded. To prevent this condition, known as thrashing, a task disabling algorithm is periodically implemented.

At predetermined intervals, the paging supervisor compares the number of Read I/O operations generated by paging requirements against predetermined high and low paging thresholds and number of reclamations (reuse of the contents of an available page) against high and low reclaim thresholds. If too much paging and reclaiming has been done during the last interval, one of the lower-priority active tasks is disabled. If more paging and reclaiming should be allowed during the next interval, a previously disabled task is reactivated.

The processing of this algorithm is performed by the Task Disablement routine (Diagram 5.19).

PAGE MANAGEMENT

Control Blocks and Tables

To efficiently manipulate pages and to maintain records of the status of page frames, the paging supervisor maintains three tables and uses the reference and change bits.

A page frame is represented by a PFTE (page frame table entry). The information in the PFTE and the settings of the reference and change bits reflect the status of a page frame. As described above (under "The Page Replacement Algorithm"), the PFTEs are the items that are moved among the available page queue, the hold page queue, and the active page queues.

Since a virtual address may exist in both real and external page storage, two tables must be kept to associate virtual addresses with their locations in the two places. The PGT (page table) is composed of 16 contiguous 2-byte PTEs (page table entries). Each entry contains the address and the availability indicators for the real storage page frame with which a particular virtual address is associated. See Figure 1-1 for an illustration of the use of a page table. One PGT exists for each segment of assigned virtual storage and can be accessed through the segment table entry for that segment.

Associated with each PGT (except those for the resident nucleus, SQA, and V=R segments) is an XPT (external page table) composed of 16 contiguous 8-byte XPTEs (external page table entries). Each XPTE contains the location of the external storage page assigned to a virtual address and the status information germain to that virtual address space.

External Storage Management

Since pages for a program need not be contiguous on external page storage devices, the paging supervisor attempts to use paging devices efficiently by assigning pages so as to minimize rotational and seek delay. Unless a specific external address is specified in the page-out request (directed page-out), the Auxiliary Storage Manager (Diagram 5.63) chooses page slots on the primary paging device that has the most pages available. A target slot is then chosen in the last cylinder used (for movable-head devices) or the next slot that will pass the read/write head (for fixed-head devices).

To further minimize paging delays, an attempt is made to use rapid-access device as primary paging devices first and to put pages on slower (designated as secondary) devices only when necessary. Migration of pages from primary to secondary devices (performed by the Migration routine, Diagram 5.62) occurs only when the number of available pages on primary devices becomes too small to keep paging efficient.

Real Storage Management

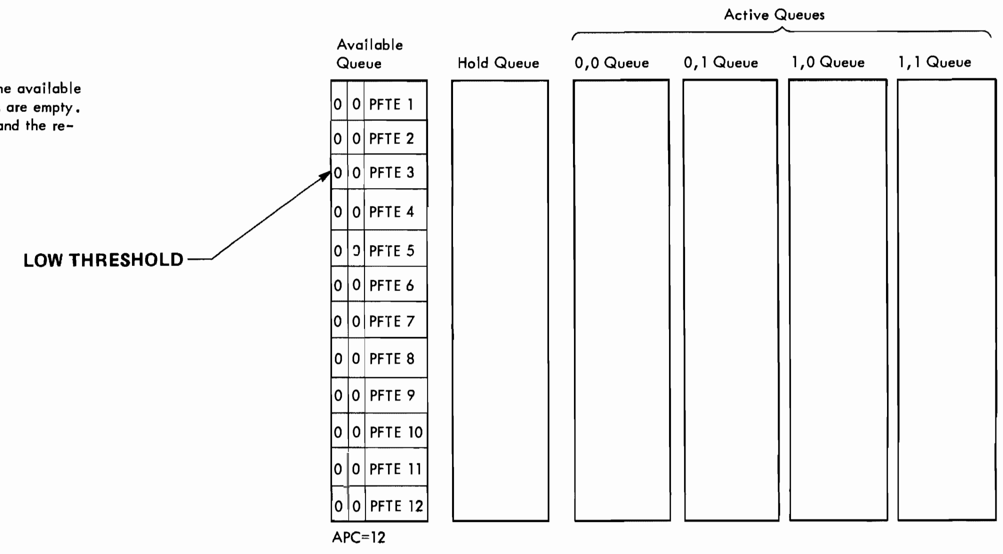
Real storage is divided into two major parts: that which is permanently allocated for supervisor use (nondynamic storage) and that which is eligible for paging (dynamic storage). Dynamic storage is further divided by a V=R line. Storage below this line is eligible for use by (but may not be exclusively used by) nonpagable (V=R, or virtual equals real) tasks. When storage below the V=R line is used for a V=R task, there is a correspondence between the virtual and real addresses.

<u>V=R Allocation</u>: A V=R region must be loaded into contiguous real storage. The paging supervisor attempts to keep a section of storage below the V=R line "nonpolluted" (not long-fixed and, if possible, not used for SQA or LSQA) so that it is available for V=R regions. Whenever a real storage page must be long-fixed or allocated to SQA or LSQA, the paging supervisor attempts to find an available page above the V=R line first. If, however, a page is needed and only pages below the V=R line are available, a V=R page is allocated.

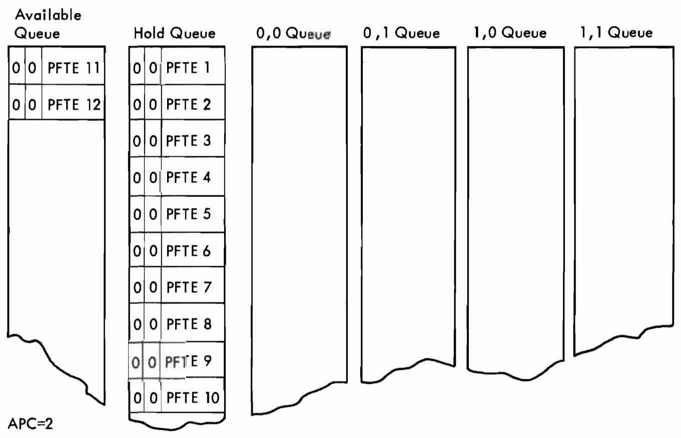
If, while a page below the V=R line is in use by one task, that in-use page is needed by a V=R task, the paging supervisor attempts to move the contents of that page to an available page above the V=R line. If that is not possible, the V=R allocation request is deferred until that page can be made available.

This series of diagrams shows the movement of PFTEs among paging supervisor queues to illustrate the operation of the page replacement algorithm. Starting with the case where all page frames are available for allocation, the diagrams follow a typical page replacement sequence.

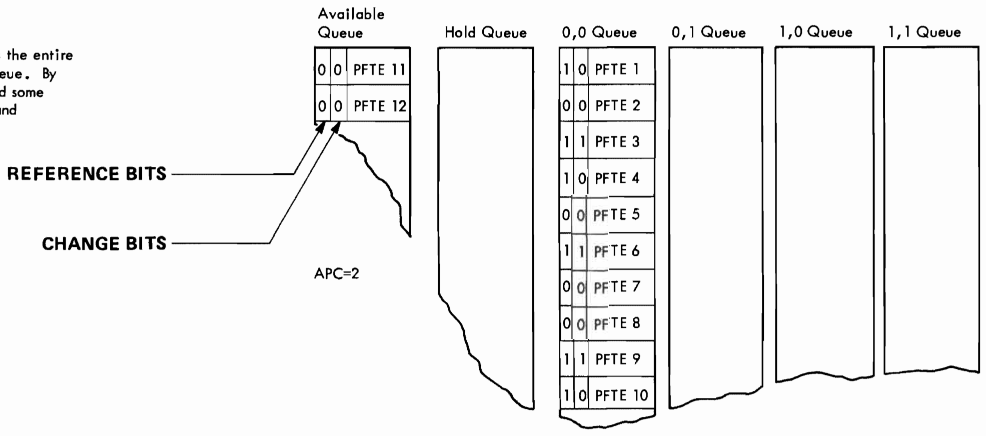
1 Initially, all PFTEs are on the available queue, the available page count (APC) is 12, and all the active queues are empty. In this example, the low threshold value is three and the replacement count is three.



2 As page frames are allocated, their PFTEs are placed on the hold queue. This step shows that ten page frames have been allocated. The allocation of page frames one through ten leaves only two PFTEs on the available queue. The next time the paging supervisor receives control, it must replenish the available queue to bring the available page count up to its high threshold value of five.

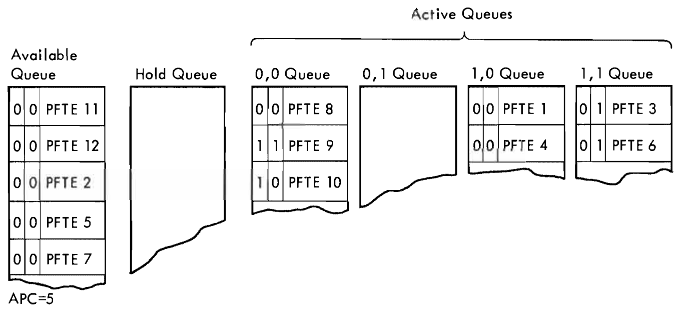


3 When the paging supervisor regains control, it places the entire hold queue on the 0,0 (unreferenced, unchanged) queue. By this time, some page frames have been referenced and some changed (as shown by the settings of the referenced and changed bits in the PFTEs).

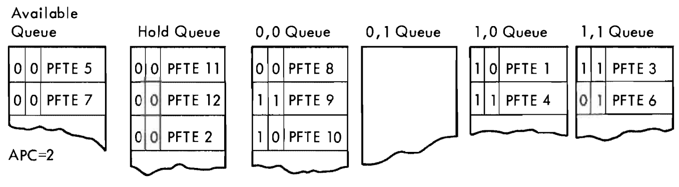


• Figure 5-1 (part 1 of 2). Movement of Page Frame Table Entries (PFTEs)

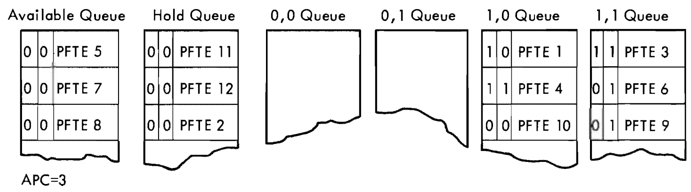
4 Next, the paging supervisor serially searches the 0,0 queue for unreferenced and unchanged page frames that it can steal for the available queue. As it examines each PFTE in turn, the paging supervisor places those PFTEs for referenced or changed page frames on the appropriate active queue and sets their reference bits to zero. It places PFTE 1 and PFTE 4 on the 1,0 (referenced, unchanged) queue, places PFTEs 2, 5, and 7 (which were unreferenced and unchanged) directly on the available queue, and sets their reference bits to zero. The addition of three PFTEs (PFTEs 2, 5, and 7) to the available queue has completed the replacement count (three). The search for page frames is ended when the paging supervisor places PFTE 7 on the available queue and the paging supervisor gives up control.



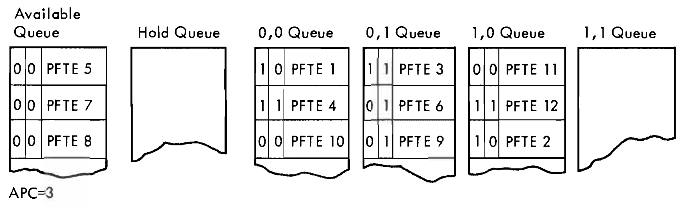
5 This step shows the PFTE queues after page frames represented by PFTEs 11, 12, and 2 have been reallocated and placed on the hold queue. Note also that the reference and change bit settings have been changed for PFTEs 1 and 4 on the 0,1 queue and for PFTE 3 on the 1,1 queue. Page frames 1 and 3 were referenced and page frame 4 was both referenced and changed after the paging supervisor gave up control following step 4. The effect of the resetting of these bits is that the paging supervisor will move these PFTEs down the active queues and thereby reduce the chance that they will be stolen. The reallocation of the page frames represented by PFTEs 11, 12, and 2 has reduced the available page count to below the low threshold value (three). Consequently, the available queue must be replenished by stealing page frames from those represented by the PFTEs on the active queues. The problem program to which the pages were allocated is given an opportunity to use those pages before the page frames that they occupy are made available for stealing. By placing PFTEs 11, 12, and 2 on the hold queue, the paging supervisor removes them from immediate consideration as candidates for stealing. The paging supervisor must steal three page frames to satisfy the replacement count. It begins by serially searching the 0,0 (unreferenced, unchanged) queue.



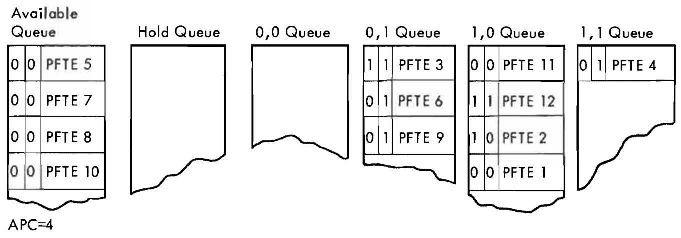
6 The paging supervisor moves PFTE 8 to the available queue because it is unreferenced and unchanged. It moves PFTEs 9 and 10 to other active queues. The replacement count has not yet been completed, so the paging supervisor must continue searching for page frames to steal to replenish the available queue. The paging supervisor continues by searching the 0,1 queue, but in this example that queue is empty. Since the paging supervisor steals from the 0,0 and 0,1 queues only, these queues must be replenished before PFTEs can be taken from them to be placed on the available queue.



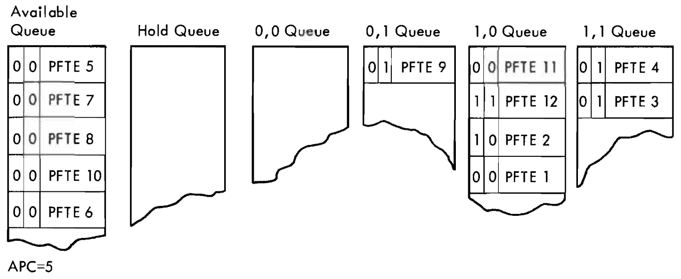
7 The paging supervisor moves the entries from the 1,0 queue to the 0,0 queue, moves the entries from the 1,1 queue to the 0,1 queue, and moves the entries from the hold queue to the 1,0 queue. Then it searches the new 0,0 queue.



8 PFTEs 1 and 4, whose bit settings changed when the paging supervisor gave up control after step 4, are moved to the 1,0 and 1,1 queues respectively. PFTE 10, being unreferenced and unchanged, is placed on the available queue. Next, the paging supervisor serially searches the 0,1 queue.



9 Page frame 3 was referenced after step 4, so the paging supervisor moves its PFTE to the 1,1 queue. Since all the PFTEs on this queue have been changed, the virtual page occupying any of the page frames represented by these PFTEs must be moved to external page storage before the PFTE can be placed on the available queue. The paging supervisor moves PFTE 6 to the available queue after moving the contents of the page frame it represents to external page storage. The replacement count has been completed and the search is ended.



• Figure 5-1 (part 2 of 2). Movement of Page Frame Table Entries (PFTEs)

Fixing Pages: It is impossible to tell when any page in the dynamic area will have to be paged out. Since, under certain conditions such as I/O buffering, it is necessary to keep a page in real storage until it is no longer needed, the ability to fix a page is provided. The fixing of a page makes that page ineligible for paging until that page is explicitly freed.

Fixing too many pages can seriously degrade real storage use and fixing pages below the V=R line can prevent a high priority V=R task from obtaining needed storage. Therefore, the number and location of fixed pages in the system is carefully controlled by the supervisor. The fixing and freeing of pages are functions of the Page Service Interface FIX/LOAD and FREE routines (Diagrams 5.21 and 5.22).

REQUEST MANAGEMENT

Since not all requests for paging services can be handled immediately, a control block must be used to represent requests for paging of, or status modifications to, pages and the storage with which they are associated. This block (the page control block, or PCB) is queued to one of ten PCB queues. Eight of the queues has an associated processor that handles the type of request represented by PCBs on that queue. (The other two queues have no processors and contain completed PCBs or PCBs for which I/O is in progress.) The queues are scanned in priority order by the Queue Scanner (Diagram 5.53) and when one is found with requests for paging supervisor operations on it, its processor is given control.

PCBs are placed on PCB queues only when the paging service they require cannot be immediately performed. When the paging task (the section of the paging supervisor that handles queued requests) is dispatched, these queues are searched and their requests fullfilled. For example, if a task requests that a virtual page be copied into real storage from external page storage (via a page fault), the PCB for this request might be queued and moved as follows:

1. A PCB is obtained from the free PCB queue or, if none is free, a new PCB is built. The PCB is then initialized to contain perinent information about the request type and to contain pointers to table entries associated with the page requested.

2. A real storage page is allocated. If one is not immediately available, the PCB is placed on the real storage allocation queue to await processing.

3. When allocation of a real storage page is complete, the PCB is moved to the page I/O initiation queue to await scheduling of the I/O operation required for the page-in.

4. When I/O operation has been scheduled, the PCB is moved to the I/O active queue to await completion of the I/O operation.

5. When the I/O operation has completed, the PCB is moved to the free PCB queue to be used for another later request.

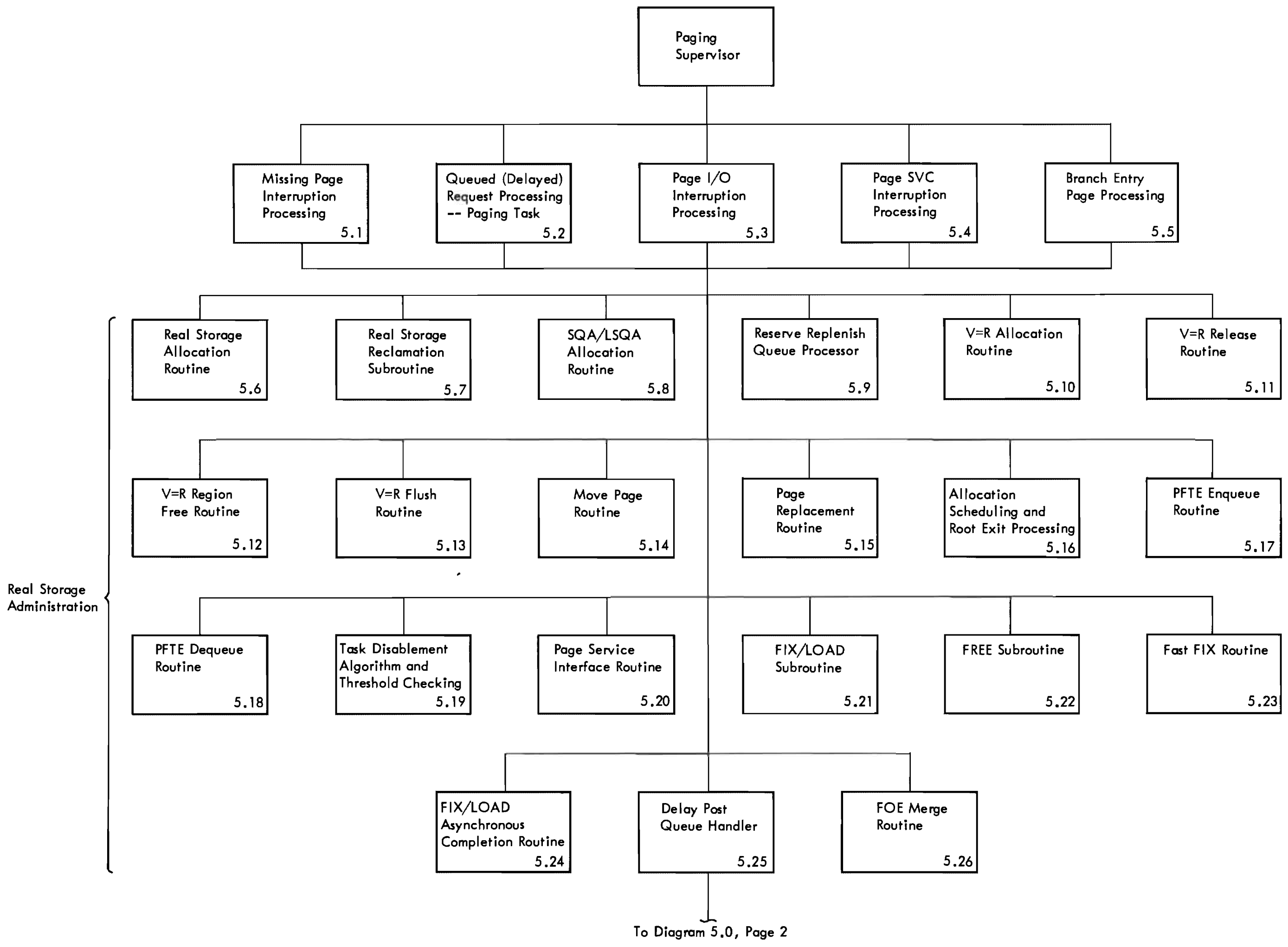
This processing is transparent to the user so that it appears to him that he has just referenced an address in real storage.

See Diagram 5.2 for a list of all PCB queues, their processors, and the flow of control after the Queue Scanner is dispatched.

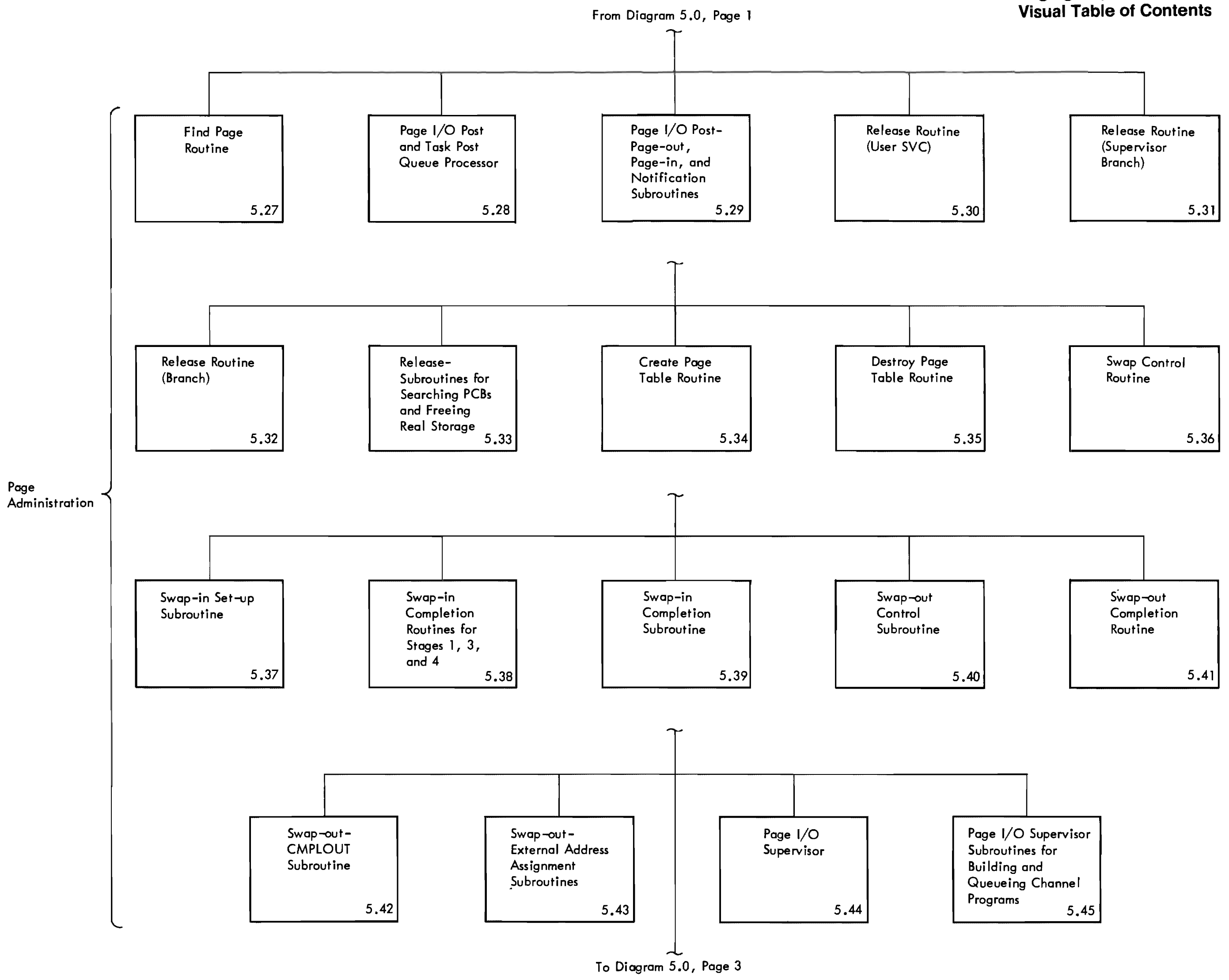
• **Diagram 5.0 (Page 1 of 3)**
Paging Supervisor
Visual Table of Contents

Index to Diagrams by Module Name for Paging Supervision

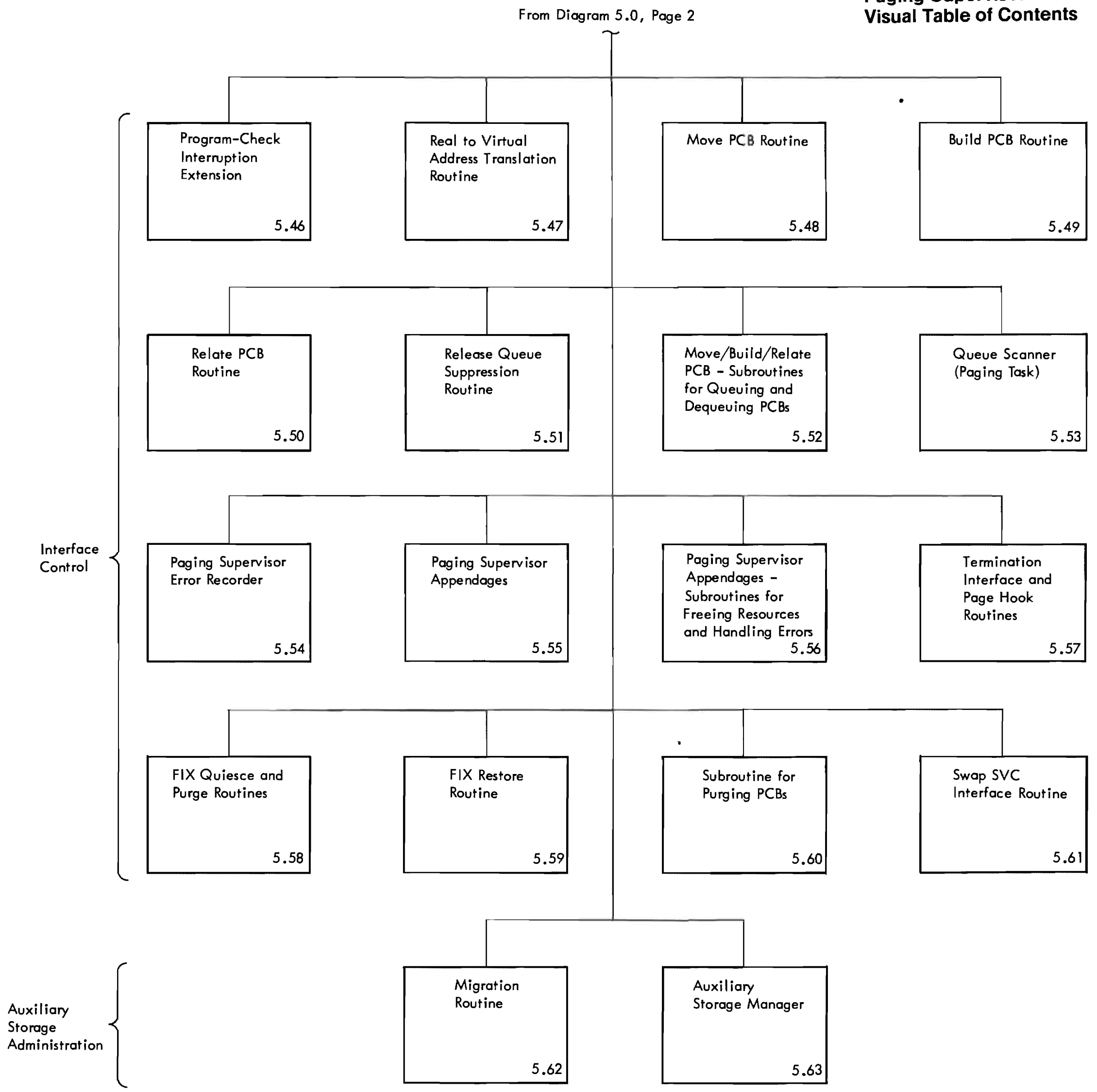
Module Name	Diagram Number(s)
IEAPALOC	5.6, 5.7
IEAPAUX	5.63
IEAPCB	5.48--5.52
IEAPCLR	5.30--5.33
IEAPFP	5.27
IEAPIOP	5.28, 5.29
IEAPIOS	5.44, 5.45
IEAPIX	5.46
IEAPMIGR	5.62
IEAPPCIA	5.55, 5.56
IEAPQS	5.53
IEAPRPLS	5.15--5.19
IEAPSER	5.54
IEAPSI	5.20--5.26
IEAPSQA	5.8, 5.9
IEAPSSVC	5.61
IEAPSWAP	5.36--5.43
IEAPTCD	5.34, 5.35
IEAPTERM	5.57--5.60
IEAPTRV	5.47
IEAPVEQR	5.10--5.14


Paging Supervisor
Missing Page Interruption Processing 5.1
Queued (Delayed) Request Processing -- Paging Task 5.2
Page I/O Interruption Processing 5.3
Page SVC Interruption Processing 5.4
Branch Entry Page Processing 5.5
Real Storage Administration
Real Storage Allocation Routine 5.6
Real Storage Reclamation Subroutine 5.7
SQA/LSQA Allocation Routine 5.8
Reserve Replenish Queue Processor 5.9
V=R Allocation Routine 5.10
V=R Release Routine 5.11
V=R Region Free Routine 5.12
V=R Flush Routine 5.13
Move Page Routine 5.14
Page Replacement Routine 5.15
Allocation Scheduling and Root Exit Processing 5.16
PFTE Enqueue Routine 5.17
PFTE Dequeue Routine 5.18
Task Disablement Algorithm and Threshold Checking 5.19
Page Service Interface Routine 5.20
FIX/LOAD Subroutine 5.21
FREE Subroutine 5.22
Fast FIX Routine 5.23
FIX/LOAD Asynchronous Completion Routine 5.24
Delay Post Queue Handler 5.25
FOE Merge Routine 5.26
To Diagram 5.0, Page 2

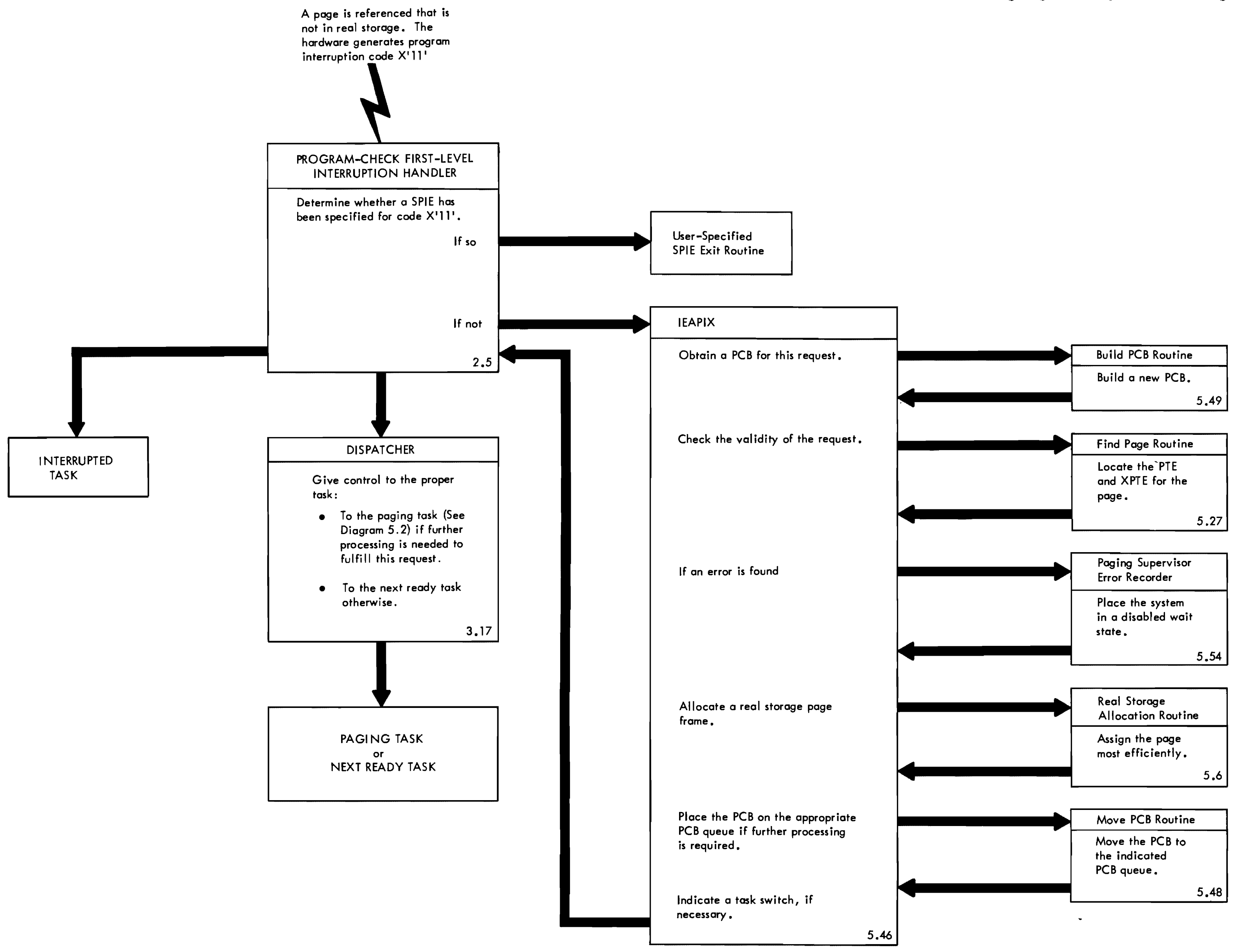
• **Diagram 5.0 (Page 2 of 3)**
Paging Supervisor
Visual Table of Contents


From Diagram 5.0, Page 1
Find Page Routine 5.27
Page I/O Post and Task Post Queue Processor 5.28
Page I/O Post-Page-out, Page-in, and Notification Subroutines 5.29
Release Routine (User SVC) 5.30
Release Routine (Supervisor Branch) 5.31
Release Routine (Branch) 5.32
Release-Subroutines for Searching PCBs and Freeing Real Storage 5.33
Create Page Table Routine 5.34
Destroy Page Table Routine 5.35
Swap Control Routine 5.36
Swap-in Set-up Subroutine 5.37
Swap-in Completion Routines for Stages 1, 3, and 4 5.38
Swap-in Completion Subroutine 5.39
Swap-out Control Subroutine 5.40
Swap-out Completion Routine 5.41
Swap-out-CMPLOUT Subroutine 5.42
Swap-out-External Address Assignment Subroutines 5.43
Page I/O Supervisor 5.44
Page I/O Supervisor Subroutines for Building and Queueing Channel Programs 5.45
Page Administration
To Diagram 5.0, Page 3

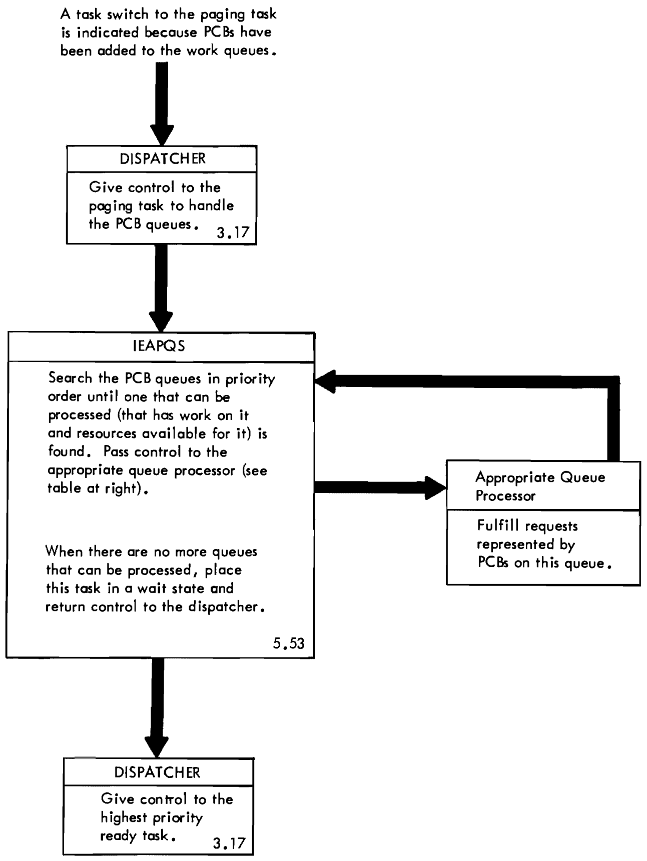
● **Diagram 5.0 (Page 3 of 3) Paging Supervisor Visual Table of Contents**


From Diagram 5.0, Page 2
Interface Control
Program-Check Interruption Extension 5.46
Real to Virtual Address Translation Routine 5.47
Move PCB Routine 5.48
Build PCB Routine 5.49
Relate PCB Routine 5.50
Release Queue Suppression Routine 5.51
Move/Build/Relate PCB - Subroutines for Queuing and Dequeuing PCBs 5.52
Queue Scanner (Paging Task) 5.53
Paging Supervisor Error Recorder 5.54
Paging Supervisor Appendages 5.55
Paging Supervisor Appendages - Subroutines for Freeing Resources and Handling Errors 5.56
Termination Interface and Page Hook Routines 5.57
FIX Quiesce and Purge Routines 5.58
FIX Restore Routine 5.59
Subroutine for Purging PCBs 5.60
Swap SVC Interface Routine 5.61
Auxiliary Storage Administration
Migration Routine 5.62
Auxiliary Storage Manager 5.63

• Diagram 5.1
Missing Page Interruption Processing

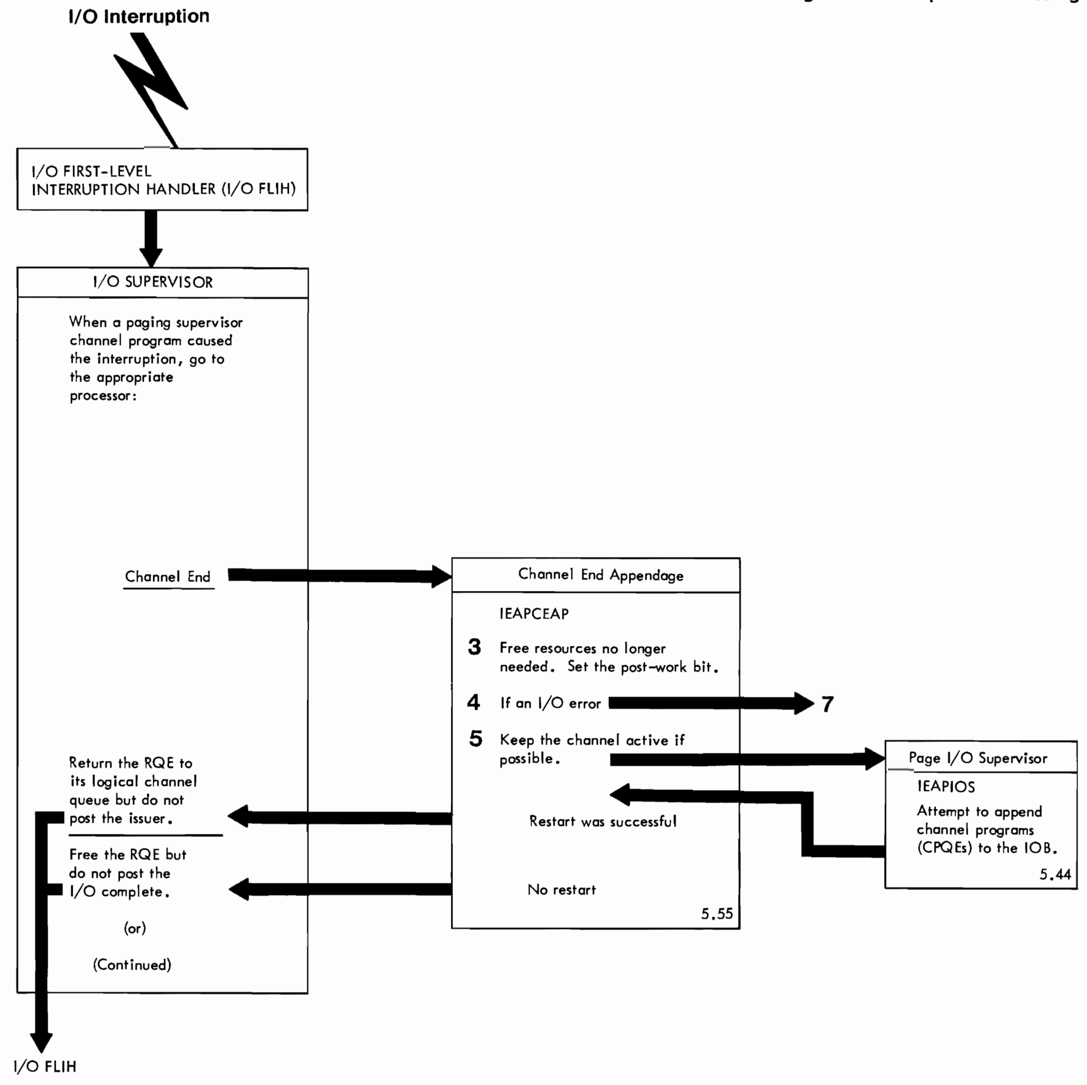

A page is referenced that is not in real storage. The hardware generates program interruption code X'11'
PROGRAM-CHECK FIRST-LEVEL INTERRUPTION HANDLER
Determine whether a SPIE has been specified for code X'11'.
If so
If not
2.5
User-Specified SPIE Exit Routine
INTERRUPTED TASK
DISPATCHER
Give control to the proper task:
• To the paging task (See Diagram 5.2) if further processing is needed to fulfill this request.
• To the next ready task otherwise.
3.17
PAGING TASK or NEXT READY TASK
IEAPIX
Obtain a PCB for this request.
Check the validity of the request.
If an error is found
Allocate a real storage page frame.
Place the PCB on the appropriate PCB queue if further processing is required.
Indicate a task switch, if necessary.
5.46
Build PCB Routine
Build a new PCB.
5.49
Find Page Routine
Locate the PTE and XPTE for the page.
5.27
Paging Supervisor Error Recorder
Place the system in a disabled wait state.
5.54
Real Storage Allocation Routine
Assign the page most efficiently.
5.6
Move PCB Routine
Move the PCB to the indicated PCB queue.
5.48

• Diagram 5.2 Queued (Delayed) Request Processing — Paging Task

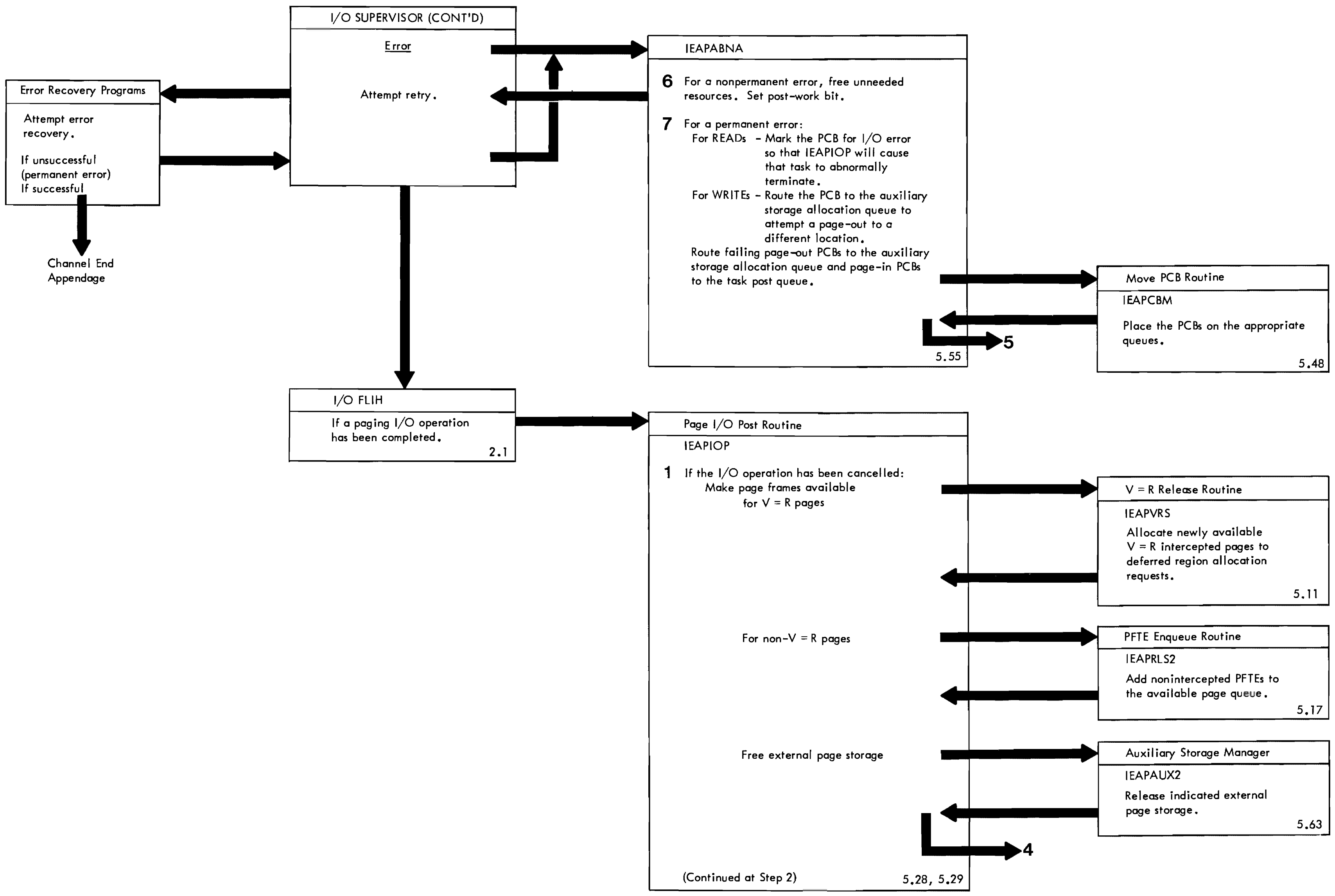

A task switch to the paging task is indicated because PCBs have been added to the work queues.
DISPATCHER
Give control to the paging task to handle the PCB queues. 3.17
IEAPQS
Search the PCB queues in priority order until one that can be processed (that has work on it and resources available for it) is found. Pass control to the appropriate queue processor (see table at right).
When there are no more queues that can be processed, place this task in a wait state and return control to the dispatcher. 5.53
Appropriate Queue Processor
Fulfill requests represented by PCBs on this queue.
DISPATCHER
Give control to the highest priority ready task. 3.17

PCB QUEUE NUMBER (by priority)	QUEUE NAME	PROCESSOR	REASON FOR PLACING PCB ON THIS QUEUE
1	Real Storage Allocation Queue	IEAPALOC (Diagram 5.6)	A real storage page was needed to satisfy a request, but none was available.
2	Auxiliary Storage Allocation Queue	IEAPAUXS (Diagram 5.63)	An external storage (auxiliary) page must be assigned for a page-out request.
3	Page I/O (Initiation) Queue	IEAPIOS (Diagram 5.44)	The I/O operations required for page-outs or page-ins must be initiated.
4	Migration Queue	IEAPMIGR (Diagram 5.62)	Pages must be moved from primary to secondary paging devices to make paging more efficient.
5	Swap Queue	IEAPSWAP (Diagram 5.36)	A swap operation was requested via SVC 115 and the Swap Interface routine (IEAPSSVC) has placed PCBs for the request on this queue.
6	Reserve Replenish Queue	IEAPSQA (Diagram 5.8)	A real storage page must be freed to replace a page reserved for emergency SQA use because a page reserved for SQA or LSQA has been used or the count of available pages reached zero and replacement could not be scheduled. Note: There is always a PCB on the queue; the above situations cause the queue to be unlocked.
7	Delayed Posting Queue	IEAPSI3 (Diagram 5.25)	A request for a page service that could not be immediately performed has now been completed, but the system was handling a disabled page fault at completion time.
8	Task Post Queue	IEAPTQP (Diagram 5.28)	A page was reclaimed in the process of a page-out, or an I/O event for a request represented by a PCB has completed. Post processing is needed but will not be initiated by an I/O interruption.
*9	I/O Active Queue		A paging I/O operation has been initiated to satisfy the request represented by this PCB.
*10	Free Queue		A request has been completely satisfied and this PCB is no longer needed. These PCBs are available for use in handling other requests.

* These queues are not checked by the queue scanner, but PCBs are routed to them by some of the queue processors.


I/O Interruption
I/O FIRST-LEVEL INTERRUPTION HANDLER (I/O FLIH)
I/O SUPERVISOR
When a paging supervisor channel program caused the interruption, go to the appropriate processor:
Channel End
Return the RQE to its logical channel queue but do not post the issuer.
Free the RQE but do not post the I/O complete.
(or)
(Continued)
I/O FLIH
Channel End Appendage
IEAPCEAP
3 Free resources no longer needed. Set the post-work bit.
4 If an I/O error
7
5 Keep the channel active if possible.
Restart was successful
No restart
5.55
Page I/O Supervisor
IEAPIOS
Attempt to append channel programs (CPQEs) to the IOB.
5.44

• **Diagram 5.3 (Page 2 of 3)**
Page I/O Interruption Processing



I/O SUPERVISOR (CONT'D)

Error

Attempt retry.

Error Recovery Programs

Attempt error recovery.

If unsuccessful (permanent error)
If successful

Channel End Appendage

IEAPABNA

6 For a nonpermanent error, free unneeded resources. Set post-work bit.

7 For a permanent error:
For READs – Mark the PCB for I/O error so that IEAPIOP will cause that task to abnormally terminate.
For WRITEs – Route the PCB to the auxiliary storage allocation queue to attempt a page-out to a different location.
Route failing page-out PCBs to the auxiliary storage allocation queue and page-in PCBs to the task post queue.

5.55

5

Move PCB Routine
IEAPCBM
Place the PCBs on the appropriate queues.
5.48

I/O FLIH
If a paging I/O operation has been completed.
2.1

Page I/O Post Routine
IEAPIOP

1 If the I/O operation has been cancelled:
Make page frames available for V = R pages

For non-V = R pages

Free external page storage

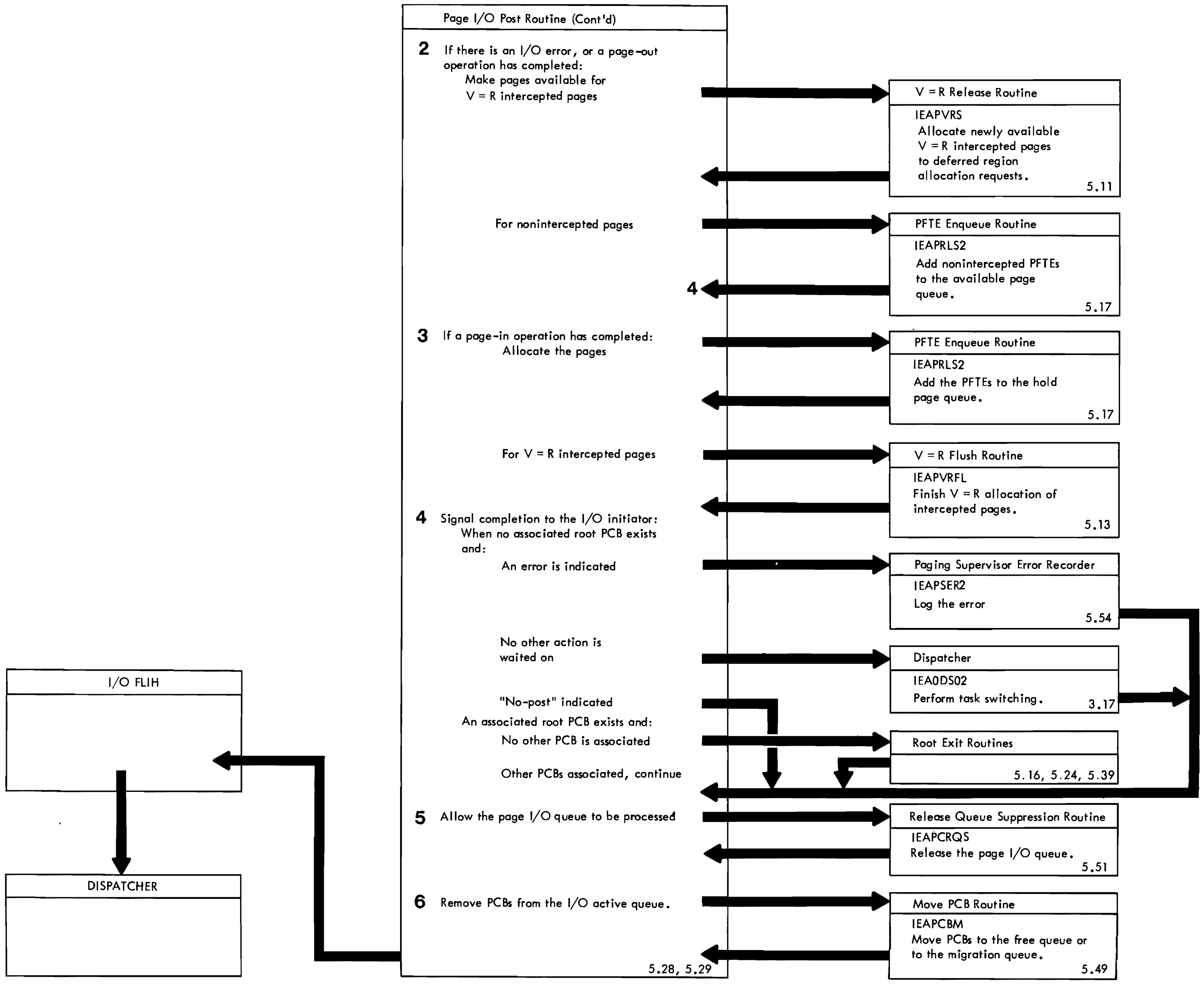
4

(Continued at Step 2) 5.28, 5.29

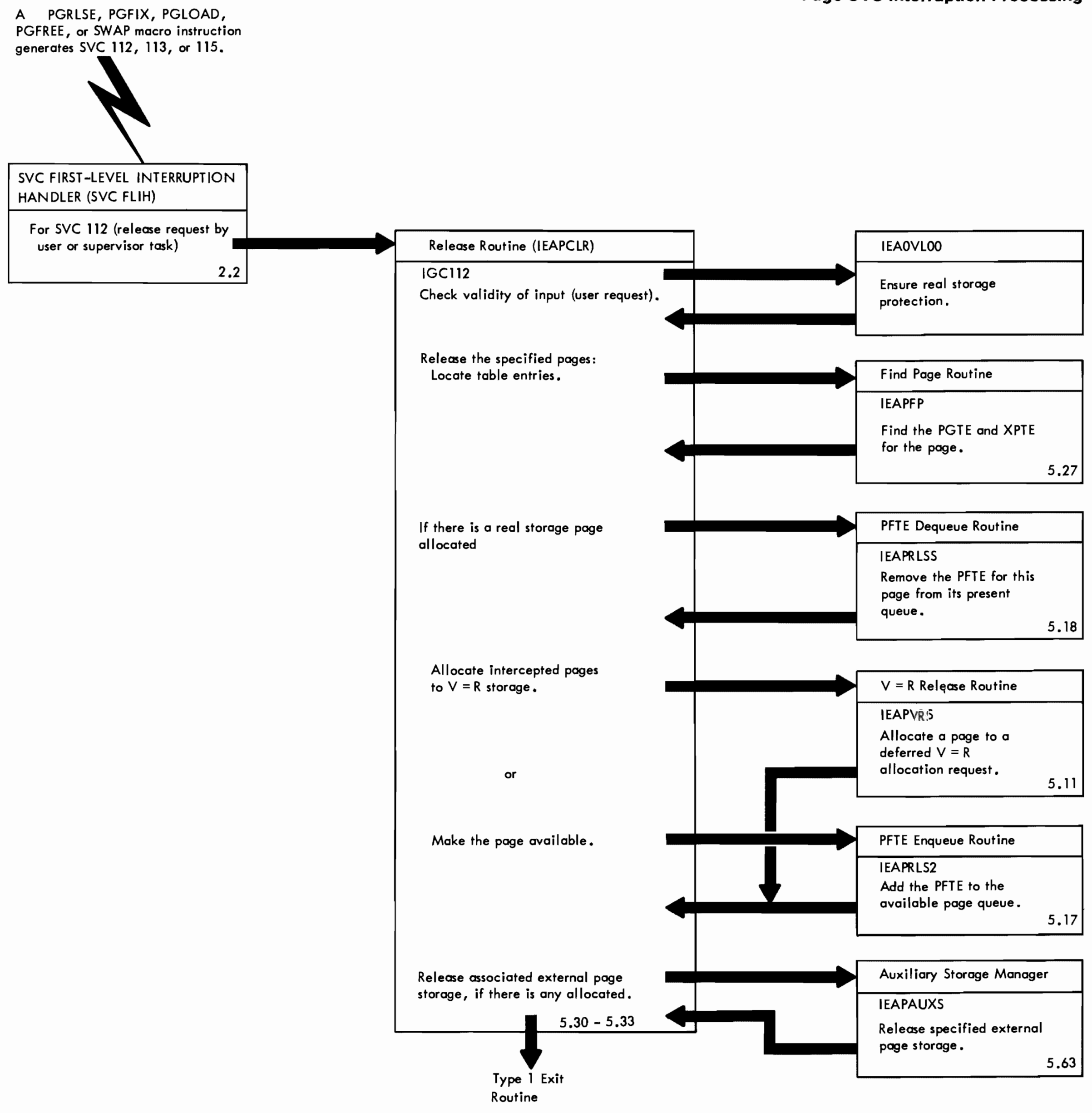
V = R Release Routine
IEAPVRS
Allocate newly available V = R intercepted pages to deferred region allocation requests.
5.11

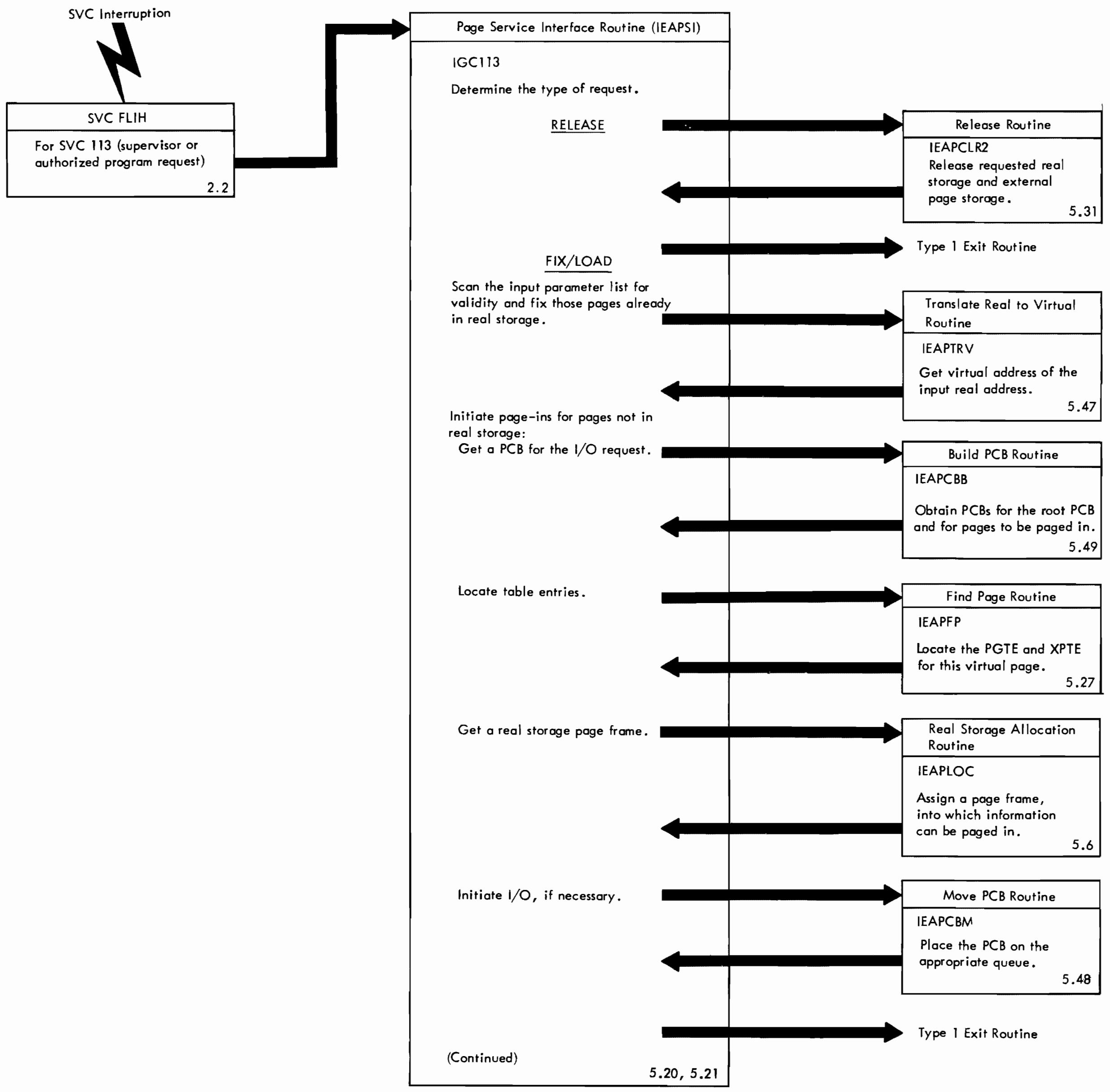
PFTE Enqueue Routine
IEAPRLS2
Add nonintercepted PFTEs to the available page queue.
5.17

Auxiliary Storage Manager
IEAPAUX2
Release indicated external page storage.
5.63

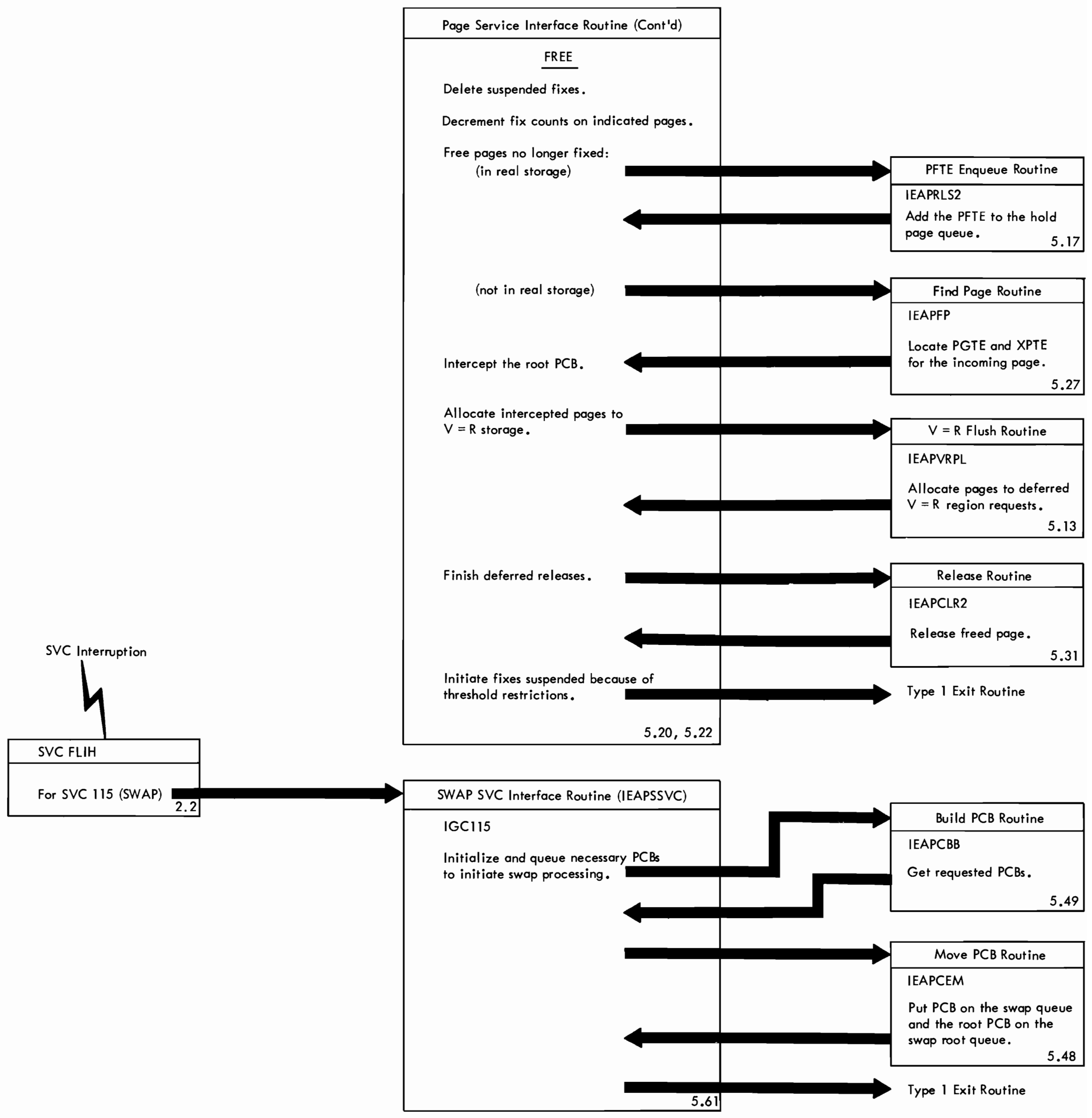

Page I/O Post Routine (Cont'd)
2 If there is an I/O error, or a page-out operation has completed: Make pages available for V = R intercepted pages
For nonintercepted pages
4
3 If a page-in operation has completed: Allocate the pages
For V = R intercepted pages
4 Signal completion to the I/O initiator: When no associated root PCB exists and: An error is indicated
No other action is waited on
"No-post" indicated
An associated root PCB exists and: No other PCB is associated
Other PCBs associated, continue
5 Allow the page I/O queue to be processed
6 Remove PCBs from the I/O active queue.
5.28, 5.29
V = R Release Routine
IEAPVRS
Allocate newly available V = R intercepted pages to deferred region allocation requests.
5.11
PFTE Enqueue Routine
IEAPRLS2
Add nonintercepted PFTEs to the available page queue.
5.17
PFTE Enqueue Routine
IEAPRLS2
Add the PFTEs to the hold page queue.
5.17
V = R Flush Routine
IEAPVRFL
Finish V = R allocation of intercepted pages.
5.13
Paging Supervisor Error Recorder
IEAPSER2
Log the error
5.54
Dispatcher
IEAODS02
Perform task switching.
3.17
Root Exit Routines
5.16, 5.24, 5.39
Release Queue Suppression Routine
IEAPCRQS
Release the page I/O queue.
5.51
Move PCB Routine
IEAPCBM
Move PCBs to the free queue or to the migration queue.
5.49
I/O FLIH
DISPATCHER

• **Diagram 5.4 (Page 1 of 3)**
Page SVC Interruption Processing


A PGRLSE, PGFIX, PGLOAD, PGFREE, or SWAP macro instruction generates SVC 112, 113, or 115.
SVC FIRST-LEVEL INTERRUPTION HANDLER (SVC FLIH)
For SVC 112 (release request by user or supervisor task)
2.2
Release Routine (IEAPCLR)
IGC112
Check validity of input (user request).
Release the specified pages: Locate table entries.
If there is a real storage page allocated
Allocate intercepted pages to V = R storage.
or
Make the page available.
Release associated external page storage, if there is any allocated.
5.30 - 5.33
Type 1 Exit Routine
IEAOVL00
Ensure real storage protection.
Find Page Routine
IEAPFP
Find the PGTE and XPTE for the page.
5.27
PFTE Dequeue Routine
IEAPRLSS
Remove the PFTE for this page from its present queue.
5.18
V = R Release Routine
IEAPVRS
Allocate a page to a deferred V = R allocation request.
5.11
PFTE Enqueue Routine
IEAPRLS2
Add the PFTE to the available page queue.
5.17
Auxiliary Storage Manager
IEAPAUXS
Release specified external page storage.
5.63


SVC Interruption
SVC FLIH
For SVC 113 (supervisor or authorized program request)
2.2
Page Service Interface Routine (IEAPSI)
IGC113
Determine the type of request.
RELEASE
FIX/LOAD
Scan the input parameter list for validity and fix those pages already in real storage.
Initiate page-ins for pages not in real storage:
Get a PCB for the I/O request.
Locate table entries.
Get a real storage page frame.
Initiate I/O, if necessary.
(Continued)
5.20, 5.21
Release Routine
IEAPCLR2
Release requested real storage and external page storage.
5.31
Type 1 Exit Routine
Translate Real to Virtual Routine
IEAPTRV
Get virtual address of the input real address.
5.47
Build PCB Routine
IEAPCBB
Obtain PCBs for the root PCB and for pages to be paged in.
5.49
Find Page Routine
IEAPFP
Locate the PGTE and XPTE for this virtual page.
5.27
Real Storage Allocation Routine
IEAPLOC
Assign a page frame, into which information can be paged in.
5.6
Move PCB Routine
IEAPCBM
Place the PCB on the appropriate queue.
5.48
Type 1 Exit Routine

• **Diagram 5.4 (Page 3 of 3)**
Page SVC Interruption Processing

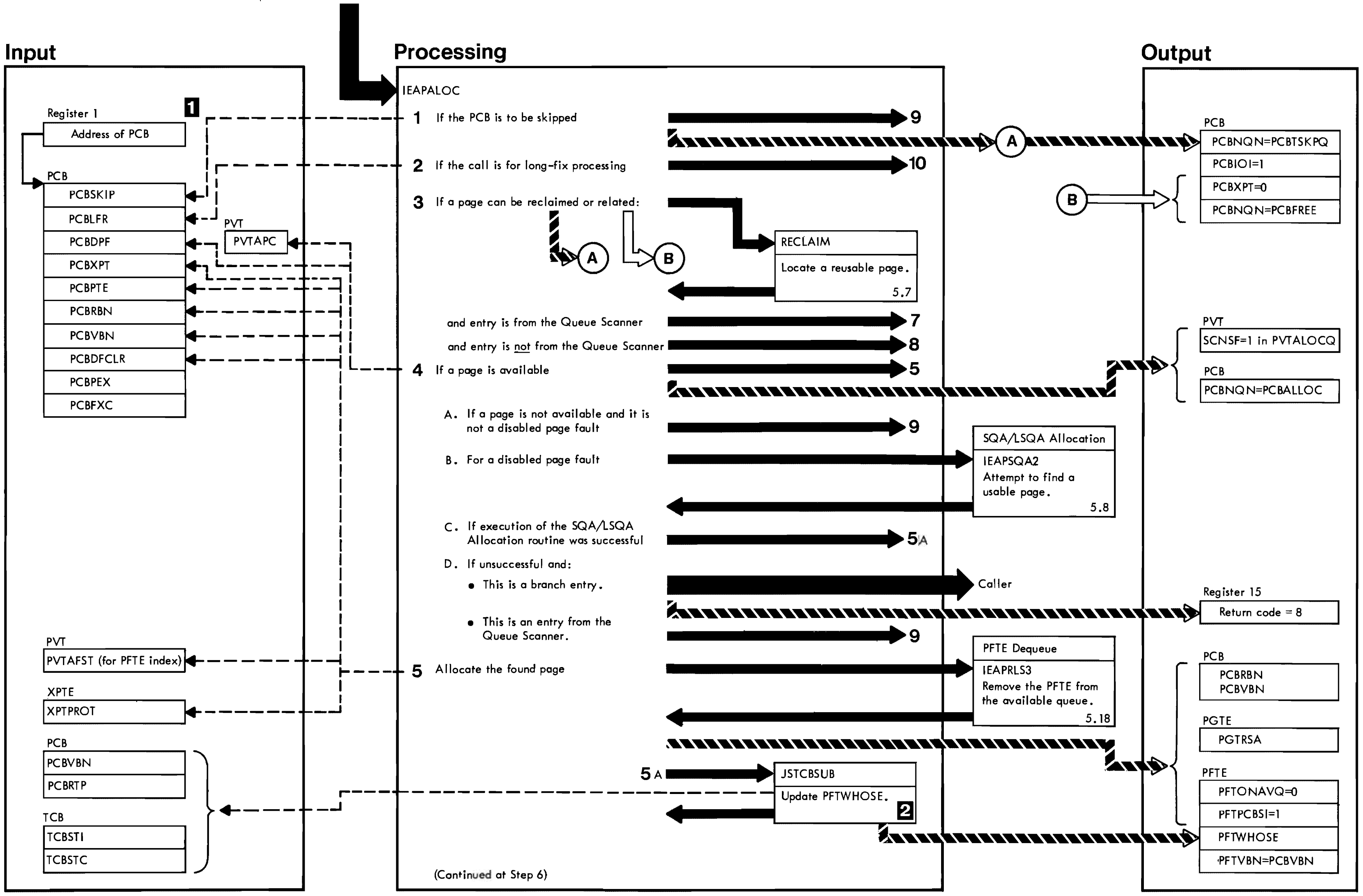

Page Service Interface Routine (Cont'd)
FREE
Delete suspended fixes.
Decrement fix counts on indicated pages.
Free pages no longer fixed:
(in real storage)
(not in real storage)
Intercept the root PCB.
Allocate intercepted pages to V = R storage.
Finish deferred releases.
Initiate fixes suspended because of threshold restrictions.
5.20, 5.22
PFTE Enqueue Routine
IEAPRLS2
Add the PFTE to the hold page queue.
5.17
Find Page Routine
IEAPFP
Locate PGTE and XPTE for the incoming page.
5.27
V = R Flush Routine
IEAPVRPL
Allocate pages to deferred V = R region requests.
5.13
Release Routine
IEAPCLR2
Release freed page.
5.31
Type 1 Exit Routine
SVC Interruption
SVC FLIH
For SVC 115 (SWAP)
2.2
SWAP SVC Interface Routine (IEAPSSVC)
IGC115
Initialize and queue necessary PCBs to initiate swap processing.
5.61
Build PCB Routine
IEAPCBB
Get requested PCBs.
5.49
Move PCB Routine
IEAPCEM
Put PCB on the swap queue and the root PCB on the swap root queue.
5.48
Type 1 Exit Routine

BRANCH ENTRY PAGE PROCESSING

CALLER	REASON	ROUTINE CALLED	ENTRY POINT AND DIAGRAM NUMBER	FUNCTION
GETMAIN	A request for SQA or LSQA space requires a new SQA page.	SQA/LSQA Allocation	IEAPSQA1, Diagram 5.8	Allocate an SQA page.
FREEMAIN	At least one whole virtual page has been freed.	Page Release	IEAPCLR2, Diagram 5.31	Release the real and external page storage associated with the freed virtual page.
GETPART	A new region is to be allocated.	Create Page Table	IEAPTCD, Diagram 5.34	Create a page table and external page table for the region. Validate segment table entries.
FREEPART	A region is to be freed.	Destroy Page Table	IEAPTCD Diagram 5.35	Free the page table and external page table for the region. Invalidate segment table entries.
	A new nonpageable (V = R) region is to be allocated.	V = R Allocation	IEAPVRAL, Diagram 5.10	Attempt to allocate a region below the V = R line.
I/O Supervisor	A paging service is needed to complete an I/O operation.	Page Service Interface	IEAPSIBR Diagram 5.20	FIX, FREE, LOAD, or RELEASE the page as requested.
I/O FLIH	Requests represented by a PCB have been completed by an I/O operation.	Page I/O Post	IEAPIOP, Diagram 5.28	Clean up resources no longer needed for this PCB, and notify the supervisor of its completion.
Program FLIH	A translation specification error (program interruption code X'12') has occurred. If the faulting task is in supervisor state, a major error is indicated. If the faulting task is in problem program state, a minor error is indicated.	Paging Supervisor Error Recorder	IEAPSER, Diagram 5.54	Major error -- inform the operator and place the system in a disabled wait state. Minor error -- inform the operator and schedule the task for abnormal termination.
Recover Management Support (RMS)	An error has been encountered in real storage.	Translate Real to Virtual	IEAPTRV, Diagram 5.47	Find the virtual address corresponding to the input real address so that RMS can access the real storage while dynamic address translation is in effect.
		Find Page	IEAPFP, Diagram 5.27	Find the page table entry and external page table entry associated with this virtual address.
	An intermittent or solid error has been encountered.	PFTE Dequeue	IEAPRLS3, Diagram 5.18	Remove the PFTE representing the bad page from its active page queue.
	An intermittent error has been encountered.	PFTE Enqueue	IEAPRLS2, Diagram 5.17	Place the PFTE representing the page in error on the available page queue.
	An intermittent error has been encountered and the page is marked for V = R interception.	V = R Release	IEAPVRS, Diagram 5.11	Allocate the newly available page to a deferred V = R region allocation request.
Error Recovery Procedures (ERPs)	An ERP that is retrying a failing paging I/O operation needs to access storage for which it has only the real address.	Translate Real to Virtual	IEAPTRV, Diagram 5.47	Find the virtual address corresponding to the input real address.
ABEND ASIR EOT	Paging for a TCB, RB, or both must be purged due to either abnormal or normal termination.	Termination Interface	IEAPTERM, Diagram 5.57	Purge resources by TCB and RB for abnormal termination. Purge resources by TCB only for normal termination.
			IEAPFIXP, Diagram 5.58	Free nonintercepted fixed pages and purge all FOEs for this TCB.
ABTERM	A task tried to schedule the paging supervisor for abnormal termination due to an invalid request, or a program check occurred in the paging supervisor causing entry to ABTERM.		PAGEHOOK, Diagram 5.57	Call IEAPSER to inform the operator and place the system in a disabled wait state.
TSO Quiesce	A TSO region is being quiesced.		IEAPFIXQ, Diagram 5.58	Quiesce all activity relating to fixes in this region.
TSO Restore	A TSO region is being restored.		IEAPFIXR, Diagram 5.59	Restore all activity relating to fixes in this region.

Diagram 5.6 (Steps 1-5) Real Storage Allocation Routine (Allocating a Page Frame)

From: Program FLIH to handle a program interruption for a page fault or a disabled page fault.
FIX to allocate or reclaim a fixed page or do long-fix processing.
Queue Scanner to process PCBs on the allocate queue.



Input

Register 1
Address of PCB

PCB
PCBSKIP
PCBLFR
PCBDPF
PCBXPT
PCBPTE
PCBRBN
PCBVBN
PCBDFCLR
PCBPEX
PCBFXC

PVT
PVTAPC

PVT
PVTAFST (for PFTE index)

XPTE
XPTPROT

PCB
PCBVBN
PCBRTP

TCB
TCBSTI
TCBSTC

Processing

IEAPALOC

1 If the PCB is to be skipped → 9

2 If the call is for long-fix processing → 10

3 If a page can be reclaimed or related:
RECLAIM — Locate a reusable page. 5.7

and entry is from the Queue Scanner → 7
and entry is not from the Queue Scanner → 8

4 If a page is available → 5

A. If a page is not available and it is not a disabled page fault → 9

B. For a disabled page fault
SQA/LSQA Allocation — IEAPSQA2 — Attempt to find a usable page. 5.8

C. If execution of the SQA/LSQA Allocation routine was successful → 5A

D. If unsuccessful and:
- This is a branch entry. → Caller
- This is an entry from the Queue Scanner. → 9

5 Allocate the found page
PFTE Dequeue — IEAPRLS3 — Remove the PFTE from the available queue. 5.18

5A JSTCBSUB — Update PFTWHOSE.

(Continued at Step 6)

Output

PCB
PCBNQN=PCBTSKPQ
PCBIOI=1
PCBXPT=0
PCBNQN=PCBFREE

PVT
SCNSF=1 in PVTALOCQ

PCB
PCBNQN=PCBALLOC

Register 15
Return code = 8

PCB
PCBRBN
PCBVBN

PGTE
PGTRSA

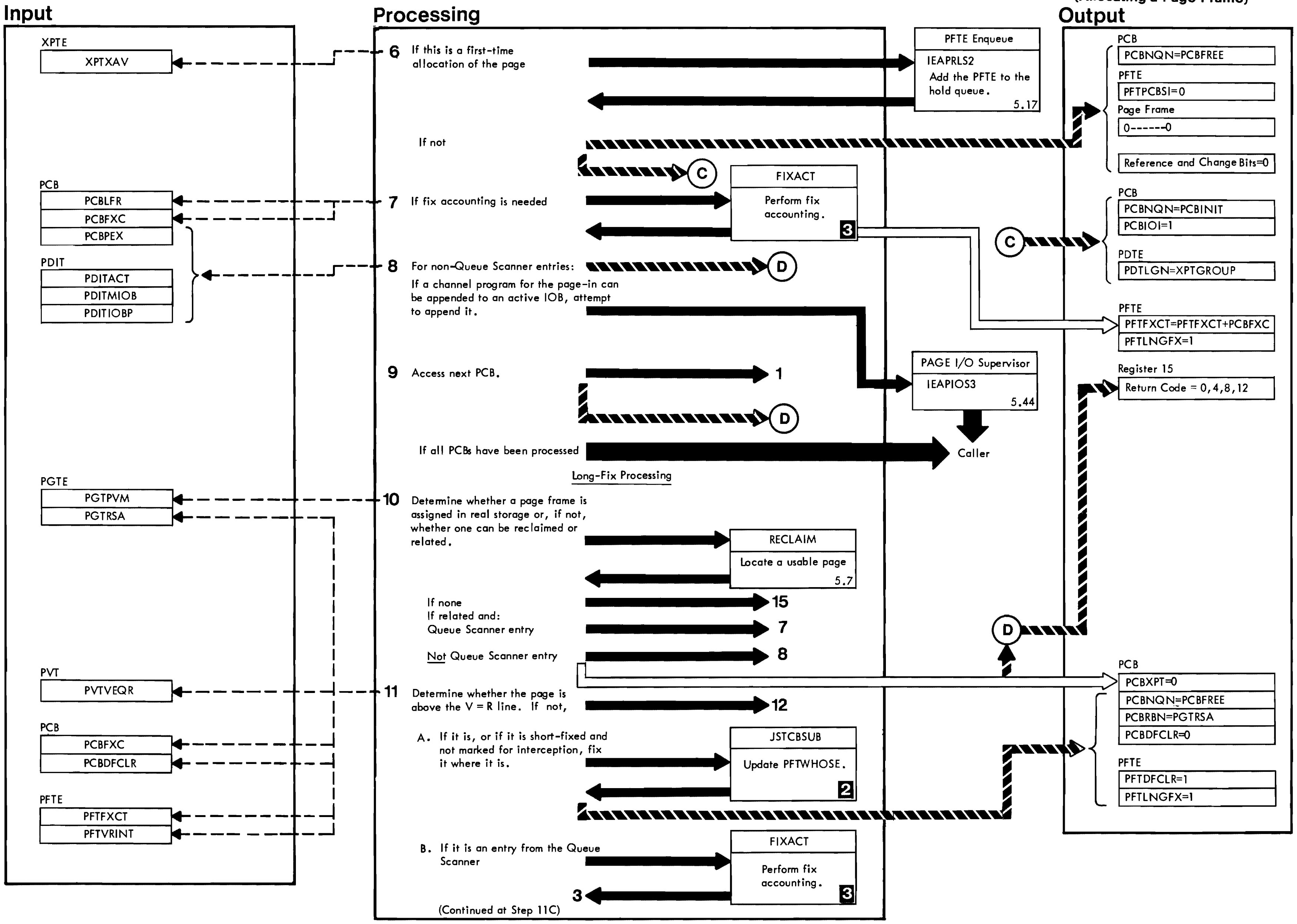
PFTE
PFTONAVQ=0
PFTPCBSI=1
PFTWHOSE
PFTVBN=PCBVBN

Diagram 5.6 (Steps 1-5) Real Storage Allocation Rcutine (Module IEAPALOC)

NOTES	ROUTINE NAME	LABEL
1 PCBIOI is set to 1 before PCBSKIP is examined. If PCBSKIP=1, route the PCB to the task post queue and check the next PCB.	IEAPALOC	QSENTRY
2 An attempt must be made to keep long-fix pages above the V=R line.		
3 • If the request was not satisfied, continue normal processing.		
• If the request was satisfied by RECLAIM, access the next PCB.		RCLGOOD1
4 If a page is available, continue normal processing. If a page is not available:		RECLFAIL
A Suppress the real storage allocation queue to prevent reentry until at least one page is available.		SQAFAIL1
B For a disabled page fault, attempt allocation from the SQA. If a page is found, continue ncrmal prccessing. If nc page can be found, route the PCB tc the real storage allocation queue.		TRYSQA
C For a PCB not on the real storage allocation queue, route it to that queue.		
D For a PCB on the real storage allocation queue attempt to satisfy the remaining requests if any with reclaimed pages.		

NOTES	ROUTINE NAME	LABEL
5 The storage key for the allocated page frame is set to the storage key found in the XPTE. Deferred release is indicated if necessary (PFTDFCLR=1 if PCBDFCLR=1). If the page was found by the SQA/LSQA Allocation routine, the call to PFTE Dequeue routine is skipped.		APCNZ
1 • Register 1 contains a positive PCB address if the entry is from the Queue Scanner.		
• Register 1 contains a negative PCB address and:		
• PCBPEX is on for a page fault or normal fix request.		
• PCBDPF is on for a disabled page fault.		
• PCBLFR is on for a long-fix request. (A request to fix one or more pages for a relatively lcng time.)		
2 JSTCBSUB sets PFTWHOSE as follows:	JSTCBSUB	
A If "no-post" is indicated (PCBNOP=1), the TCB addressed by this PCB or its Root PCB cannot be accessed. Therefore, PFTWHOSE is set to 0.		
B If PCBTCF=1, the TCB address is in PCBRTP of the PCB. If PCBTCF=0, the TCB address is in the PCBRTCB field of the root PCB pointed to by PCBRTP.		
C If TCBSTI=0, PFTWHOSE is set to 0.		
D If the page frame teing allocated falls within the region associated with the current TCB, PFTWHOSE is set to TCBJSTCB. Otherwise, it is set to 0.		

• **Diagram 5.6 (Steps 6-11B) Real Storage Allocation Routine (Allocating a Page Frame)**



6 If this is a first-time allocation of the page

If not

7 If fix accounting is needed

8 For non-Queue Scanner entries:

If a channel program for the page-in can be appended to an active IOB, attempt to append it.

9 Access next PCB.

If all PCBs have been processed

Long-Fix Processing

10 Determine whether a page frame is assigned in real storage or, if not, whether one can be reclaimed or related.

If none

If related and:

Queue Scanner entry

Not Queue Scanner entry

11 Determine whether the page is above the V = R line. If not,

A. If it is, or if it is short-fixed and not marked for interception, fix it where it is.

B. If it is an entry from the Queue Scanner

(Continued at Step 11C)

Diagram 5.6 (Steps 6-11B) Real Storage Allocation Routine (Module IEAPALOC)

NOTES	ROUTINE NAME	LABEL
6 If the page has never been pagedout (XPTXAV=0, that is it is not on external page storage), the PGTE is validated and the page is cleared to destroy any secured data in that page frame. If the page exists on external page storage, a page-in is necessary.	IEAPALOC	PAGECOM
7 If the Queue Scanner was the caller and the fix count in the PCB is greater than zero, fix accounting must be performed. If PCBLFR=1, PFTLNGFX is set to 1.		NOTDFCLR
8 If the page has previously been paged out, and the device is active (PDITACT=1), and: • The call is for a paging exception, or • The call is for FIX and it is not a Queue Scanner entry, the routine attempts to append the channel program to one already running on the device.		TESTACT
If the page has not previously been paged out, the call was for other than a paging exception, or the device was not active, or upon return from IEAPIOS3, return is made to the caller.		APPENDON
9 Return code meanings are: 0 - Page is immediately available. 4 - I/O must be performed. 8 - Unsuccessful. 12 - Related I/O must be performed.		CHECKAPC
10		LFIXREQ
11 After the page is fixed, processing continues as though the request was satisfied with a reclaimed page. If deferred release is indicated (PCBDFCLR=1), the flag is reset to zero and PFTDFCLR is set to 1. Long-fix is indicated (PFTLNGFX=1) if the caller was the Queue Scanner.		PGVALID
3 The number of fixes represented by this PCB is recorded in the PFTE.	FIXACT	

Diagram 5.6 (Steps 11C-15)
Real Storage Allocation Routine
(Allocating a Page Frame)

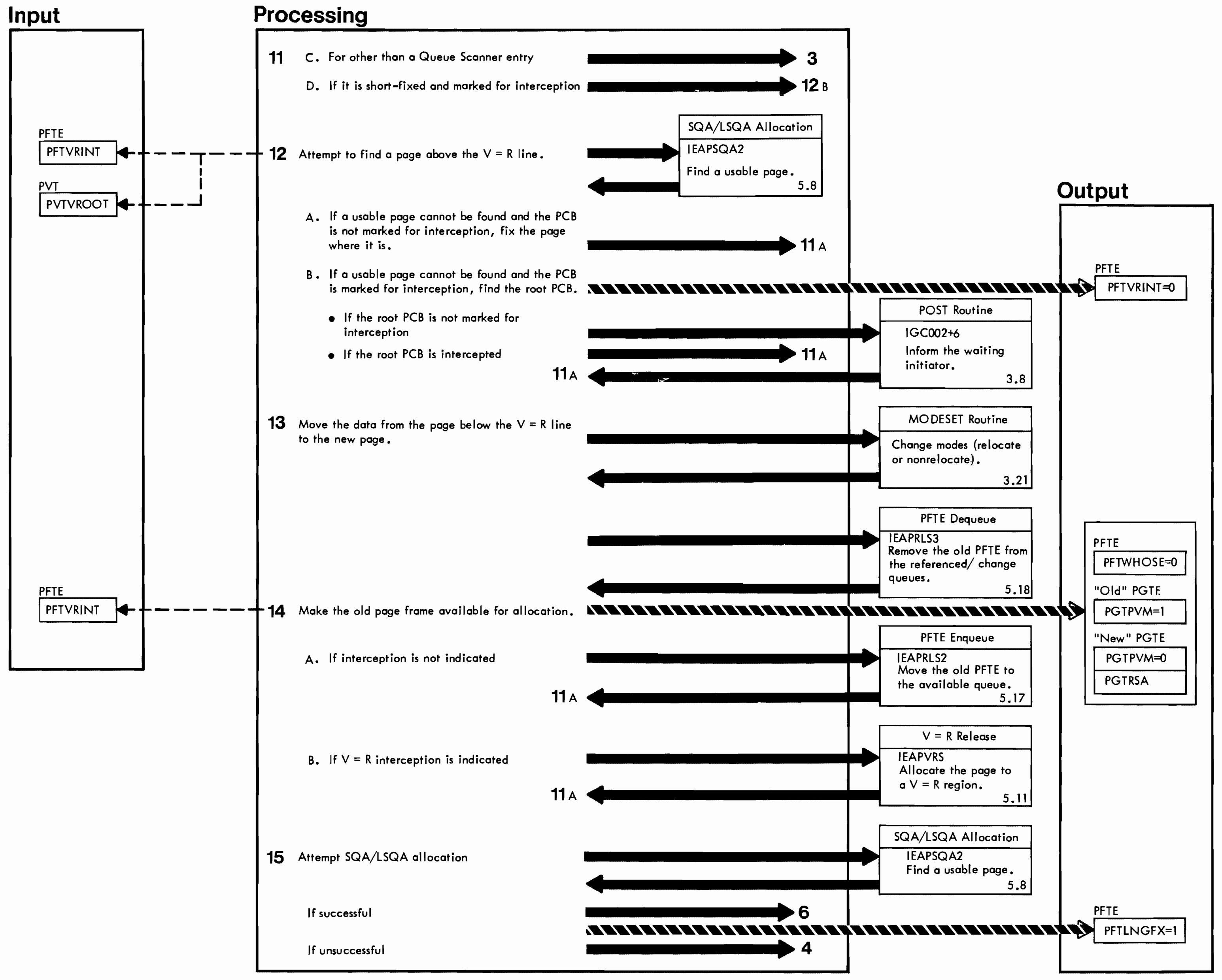
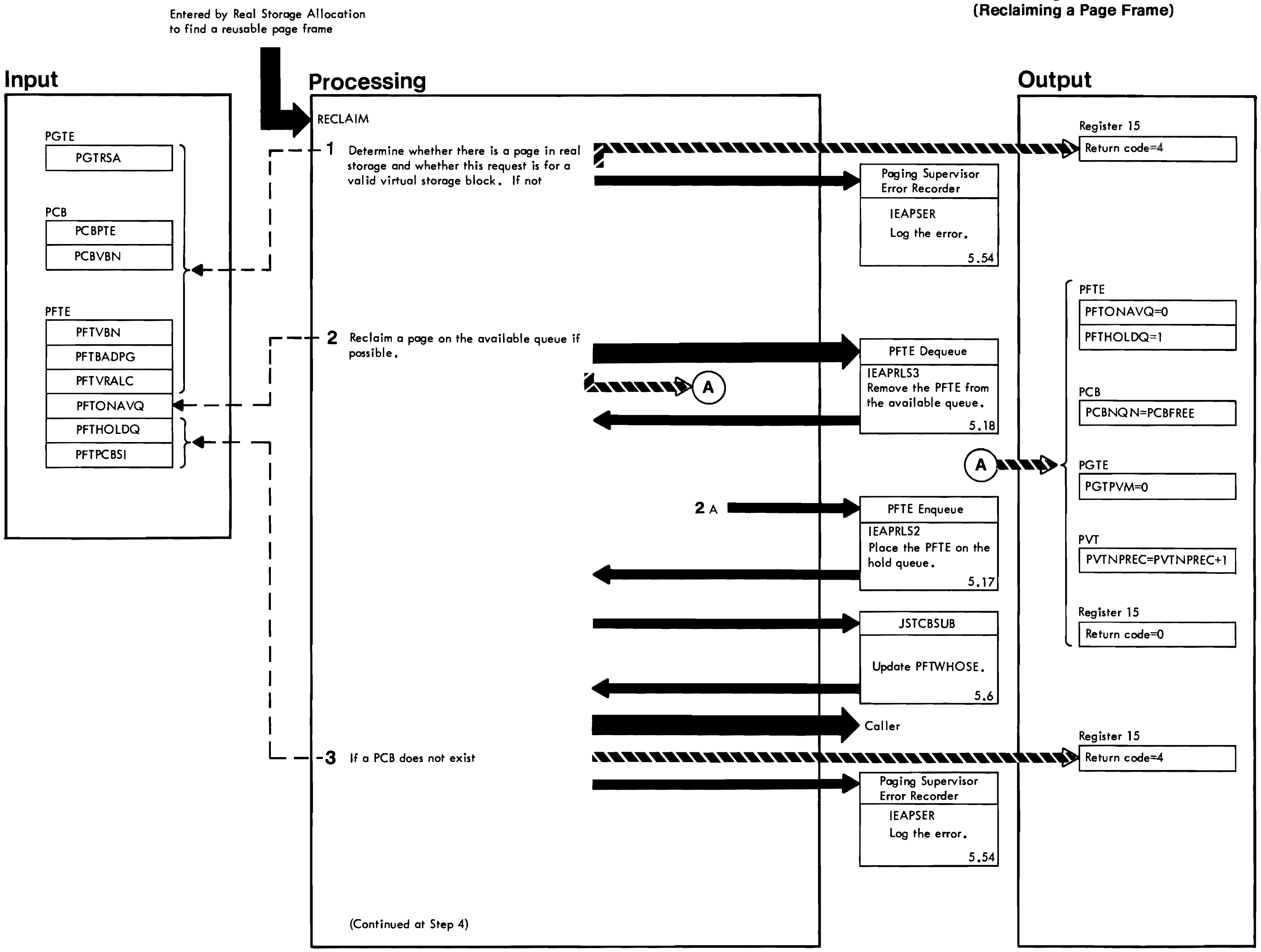

Input
Processing
Output
PFTE
PFTVRINT
PVT
PVTVROOT
PFTE
PFTVRINT
11 C. For other than a Queue Scanner entry
3
D. If it is short-fixed and marked for interception
12 B
12 Attempt to find a page above the V = R line.
SQA/LSQA Allocation
IEAPSQA2
Find a usable page.
5.8
A. If a usable page cannot be found and the PCB is not marked for interception, fix the page where it is.
11 A
B. If a usable page cannot be found and the PCB is marked for interception, find the root PCB.
PFTE
PFTVRINT=0
● If the root PCB is not marked for interception
POST Routine
IGC002+6
Inform the waiting initiator.
3.8
● If the root PCB is intercepted
11 A
11 A
13 Move the data from the page below the V = R line to the new page.
MODESET Routine
Change modes (relocate or nonrelocate).
3.21
PFTE Dequeue
IEAPRLS3
Remove the old PFTE from the referenced/ change queues.
5.18
14 Make the old page frame available for allocation.
PFTE
PFTWHOSE=0
"Old" PGTE
PGTPVM=1
"New" PGTE
PGTPVM=0
PGTRSA
A. If interception is not indicated
PFTE Enqueue
IEAPRLS2
Move the old PFTE to the available queue.
5.17
11 A
B. If V = R interception is indicated
V = R Release
IEAPVRS
Allocate the page to a V = R region.
5.11
11 A
15 Attempt SQA/LSQA allocation
SQA/LSQA Allocation
IEAPSQA2
Find a usable page.
5.8
If successful
6
PFTE
PFTLNGFX=1
If unsuccessful
4

Diagram 5.6 (Steps 11C-15) Real Storage Allocation Routine (Module IEAPALOC)

NOTES	ROUTINE NAME	LABEL
12 If the page is marked for interception, the waiting initiator is posted with a code of 16 to indicate that the region represented by this root PCB cannot be completed.	IEAPALOC	
13 The page below the V=R line ("old") is invalidated, and the one above the line ("new") is validated.		
14		NOTON
15 If unsuccessful, no pages are available and this PCB must be handled again at the next entry to this routine. If successful, normal allocation continues as if this was a request for other than a long-fix, except that long-fix is indicated (PFTLNGFX=1) if the caller was the Queue Scanner.		LFIXREQ

• **Diagram 5.7 (Steps 1-3)**
Real Storage Reclamation Subroutine
(Reclaiming a Page Frame)



1 Determine whether there is a page in real storage and whether this request is for a valid virtual storage block. If not

2 Reclaim a page on the available queue if possible.

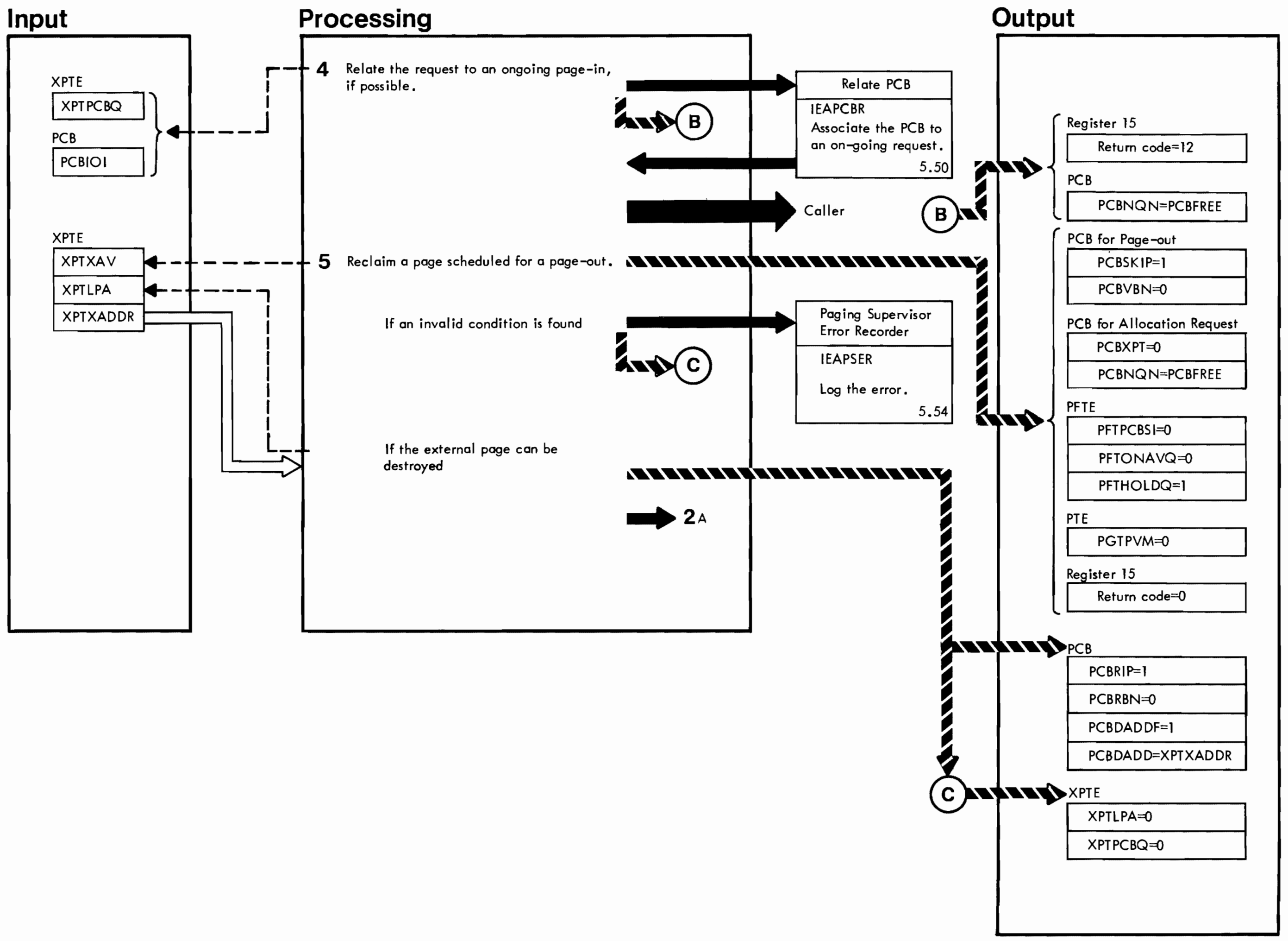
3 If a PCB does not exist

(Continued at Step 4)

• Diagram 5.7 (Steps 1-3) Real Storage Reclamation Subroutine (Module IEAPALOC)

NOTES	ROUTINE NAME	LABEL
1 If the real storage address (PGTRSA) equals zero, the virtual address is not associated with a page frame. If the virtual block numbers (virtual addresses associated with pages in real storage or in the process of being paged out) in the PFTE and PCB do not point to the same block, the page has been marked nonallocatable by RMS, or the page is allocated to a V=R region, the request is considered nonreclaimable. After issuing a console message with the appropriate code (0100,0101, 0102), IEAPSER puts the system in a disabled wait state.	RECLAIM	
2 If the available bit is on in the PFTE, the page frame contained the data, but the page frame has been released and made available.		RECLOK
3 It is first determined whether the page table entry has already been validated. After issuing a console message with the appropriate code (0100,0101,0102), IEAPSER puts the system in a disabled wait state.		PSER1

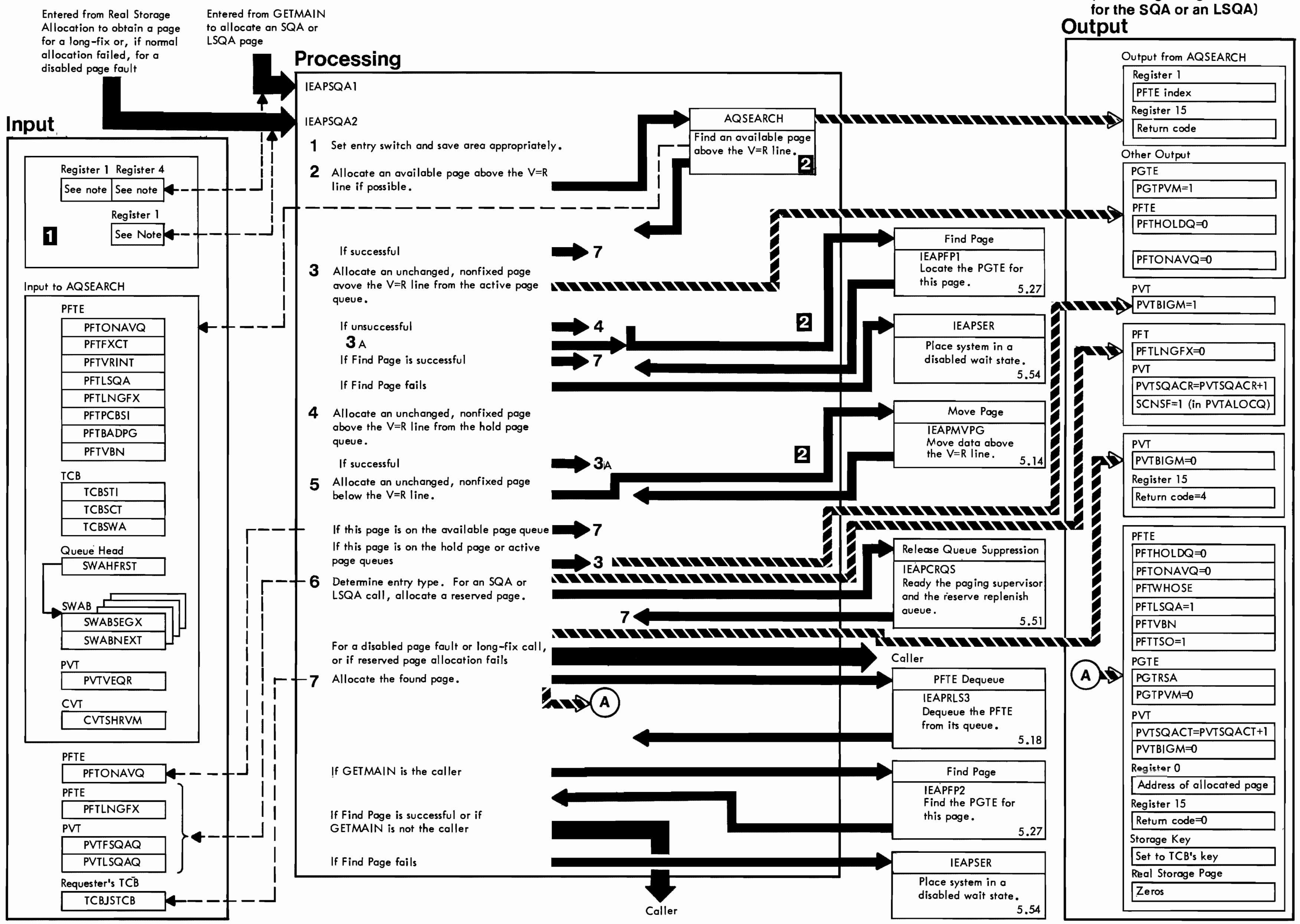
• **Diagram 5.7 (Steps 4-5)**
Real Storage Reclamation Subroutine
(Reclaiming a Page Frame)



• Diagram 5.7 (Steps 4-5) Real Storage Reclamation Subroutine (Module IEAPALOC)

NOTES	ROUTINE NAME	LABEL
4 If the page is on its way into real storage on behalf of another task, the current PCB is related to the active PCB so that both PCBs will be posted when the page is in real storage.	RECLAIM	QLOOP
5 If the page to be reclaimed is scheduled for a page out and if XPTXAV=1, XPTXAV and XPTXADDR are set to zeros.		LPAON
IEAPSER puts the system in a disabled wait state.		

• Diagram 5.8
SQA/LSQA Allocation Routine
(Allocating a Page Frame
for the SQA or an LSQA)



Entered from Real Storage Allocation to obtain a page for a long-fix or, if normal allocation failed, for a disabled page fault

Entered from GETMAIN to allocate an SQA or LSQA page

Input

Processing

IEAPSQA1

IEAPSQA2

1 Set entry switch and save area appropriately.

2 Allocate an available page above the V=R line if possible.

If successful → 7

3 Allocate an unchanged, nonfixed page avove the V=R line from the active page queue.

If unsuccessful → 4

3A

If Find Page is successful → 7

If Find Page fails

4 Allocate an unchanged, nonfixed page above the V=R line from the hold page queue.

If successful → 3A

5 Allocate an unchanged, nonfixed page below the V=R line.

If this page is on the available page queue → 7

If this page is on the hold page or active page queues → 3

6 Determine entry type. For an SQA or LSQA call, allocate a reserved page.

For a disabled page fault or long-fix call, or if reserved page allocation fails

7 Allocate the found page.

If GETMAIN is the caller

If Find Page is successful or if GETMAIN is not the caller

If Find Page fails

AQSEARCH
Find an available page above the V=R line.
2
Find Page
IEAPFP1
Locate the PGTE for this page.
5.27
2
IEAPSER
Place system in a disabled wait state.
5.54
Move Page
IEAPMVPG
Move data above the V=R line.
5.14
2
Release Queue Suppression
IEAPCRQS
Ready the paging supervisor and the reserve replenish queue.
5.51
7
Caller
PFTE Dequeue
IEAPRLS3
Dequeue the PFTE from its queue.
5.18
A
Find Page
IEAPFP2
Find the PGTE for this page.
5.27
IEAPSER
Place system in a disabled wait state.
5.54
Caller

Output

• Diagram 5.8 SQA/LSQA Allocation Routine (Module IEAPSQA)

NOTES	ROUTINE NAME	LABEL
1 The "IEAPSQ1 in control" bit is set at entry and turned off at all exits.	IEAPSQA1 IEAPSQA2	IEAPSQA1 IEAPSQA2
2		APREPSRH
3		ANEXT
4		ATSTHOLD
5 If the "alternate" switch was not set by AQSEARCH (no page exists below the V=R line either), control goes to Step 6. No data is saved from the input PFTE.		
6 The SQA reserve replenish queue is empty if PVTFSQAR=0.		
Prepare the reserve replenish queue and suppress the real storage allocation queue (SCNSF=1 in PVTALOCQ).		ATSTENTR
7 If the request is for LSQA, PFTWHOSE is set to the requester's TCB address; otherwise it is set to zero.		APGONAVL
PFTTSO is set to 1 only if GETMAIN is the caller and the TCB address is a TSO TCB.		APFTINIT
1 Register settings upon entry:		
From GETMAIN: Register 1 - Negative virtual address for SQA Positive virtual address for LSQA Register 4 - Address of requester's TCB	IEAPSQA1	
From Real Storage Allocation: Register 1 - Negative virtual address for disabled page fault Positive virtual address for long-fix	IEAPSQA2	
2 The Queue Search subroutine searches the queues of PFTEs for a page frame which is not LSQA (unless allocated to a TSO task), changed, fixed, scheduled for use (has a PCB assigned), or restricted from selection. A page is restricted if: • It is marked for interception. • It is in LPA virtual storage. • It is in the input TCB's region. • It is in an SWA segment. • It is in the V=R region. If one is found, the PFTE index is put in register 1 and control is returned to the main routine with a return code of 0. If one could not be found, control is returned with a code of 4. If one was found that meets all but the last requirement (the page is in the V=R region), its address is saved, an "alternate" switch is set, and control is returned with a code of 4.	AQSEARCH	

Diagram 5.9
Reserve Replenish Queue Processor

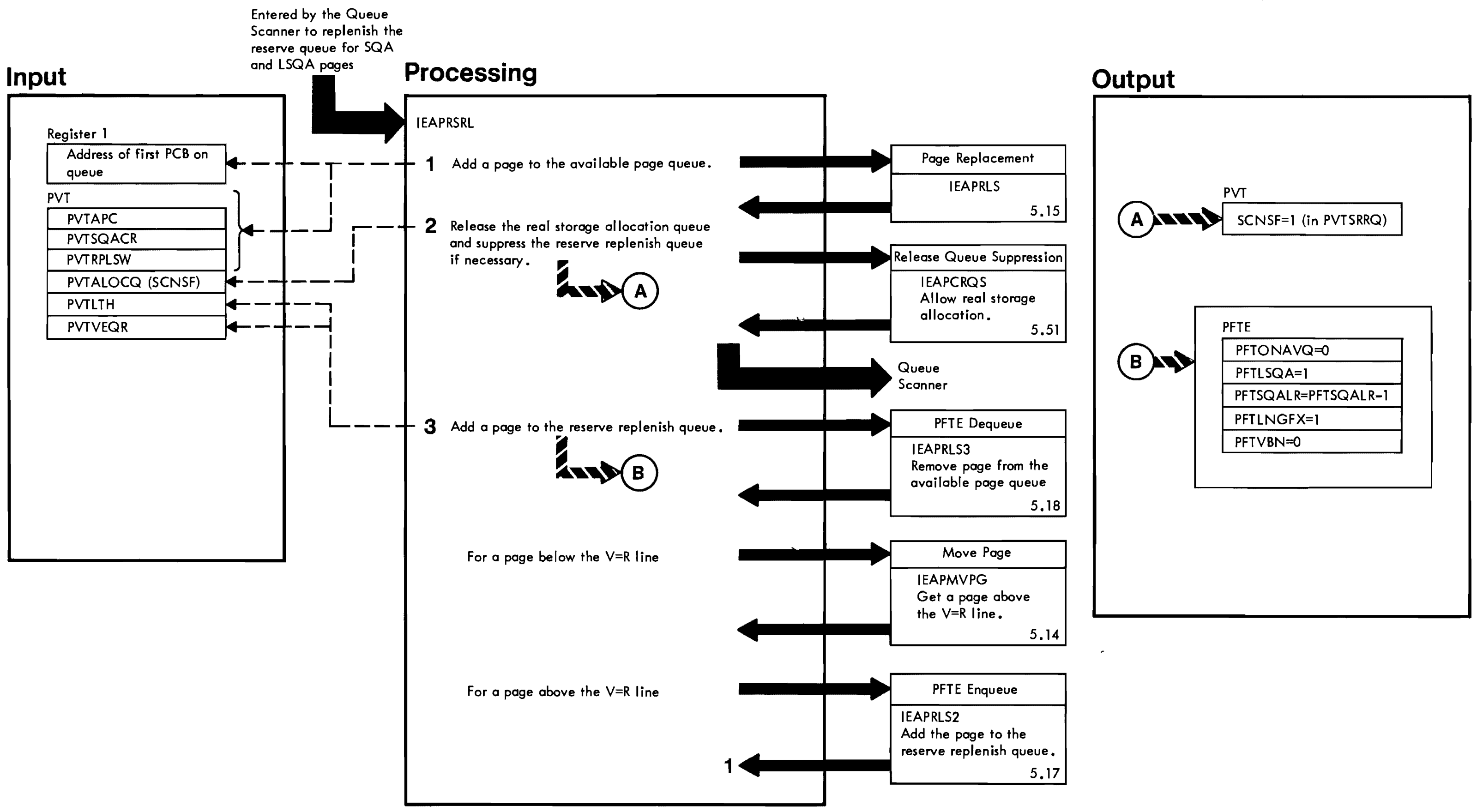
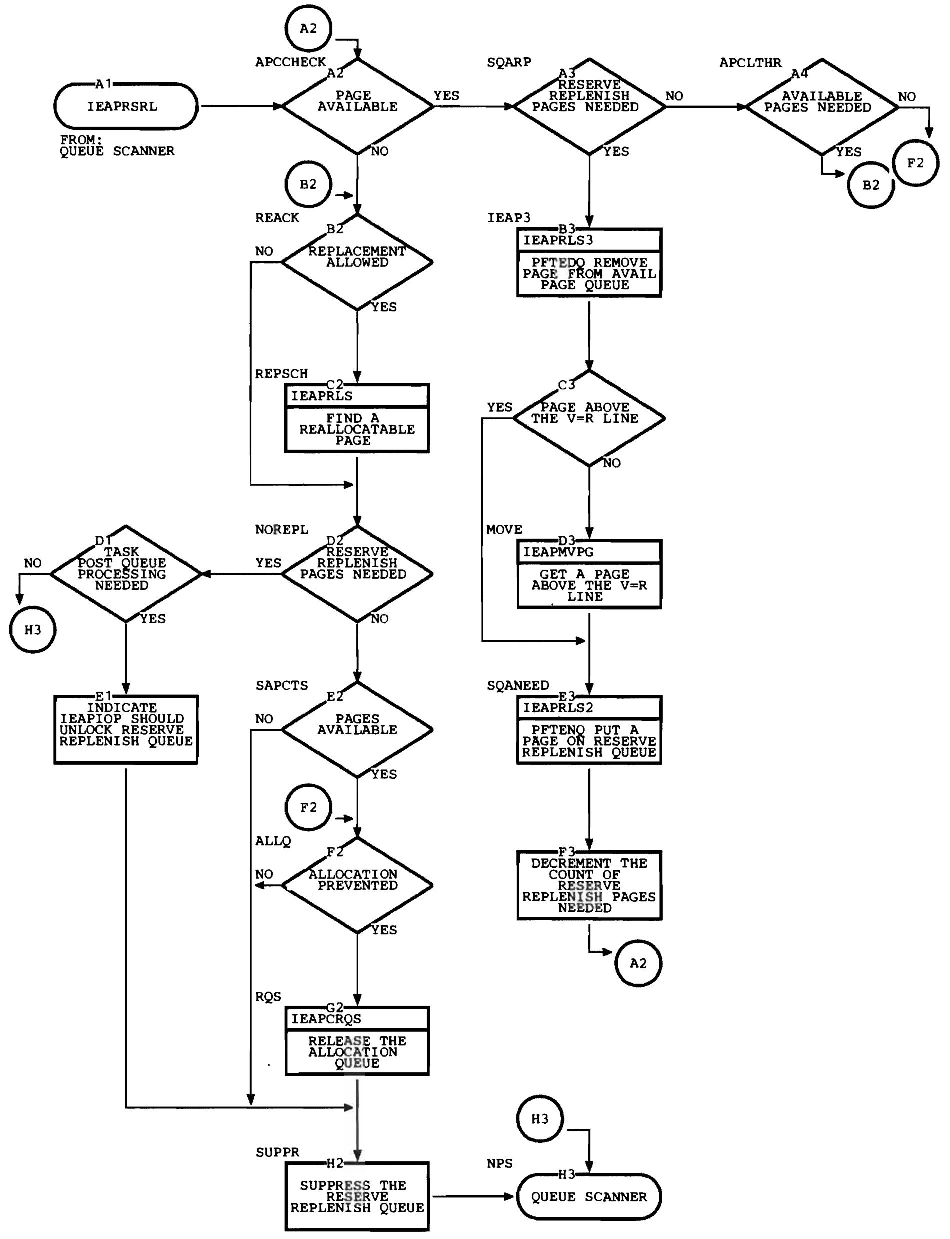

Input
Register 1
Address of first PCB on queue
PVT
PVTAPC
PVTSQACR
PVTRPLSW
PVTALOCQ (SCNSF)
PVTLTH
PVTVEQR
Entered by the Queue Scanner to replenish the reserve queue for SQA and LSQA pages
Processing
IEAPRSRL
1 Add a page to the available page queue.
2 Release the real storage allocation queue and suppress the reserve replenish queue if necessary.
A
Queue Scanner
3 Add a page to the reserve replenish queue.
B
For a page below the V=R line
For a page above the V=R line
1
Page Replacement
IEAPRLS
5.15
Release Queue Suppression
IEAPCRQS
Allow real storage allocation.
5.51
PFTE Dequeue
IEAPRLS3
Remove page from the available page queue
5.18
Move Page
IEAPMVPG
Get a page above the V=R line.
5.14
PFTE Enqueue
IEAPRLS2
Add the page to the reserve replenish queue.
5.17
Output
A
PVT
SCNSF=1 (in PVTSRRQ)
B
PFTE
PFTONAVQ=0
PFTLSQA=1
PFTSQALR=PFTSQALR-1
PFTLNGFX=1
PFTVBN=0

Diagram 5.9 Reserve Replenish Queue Processor (Module IEAPSQA)

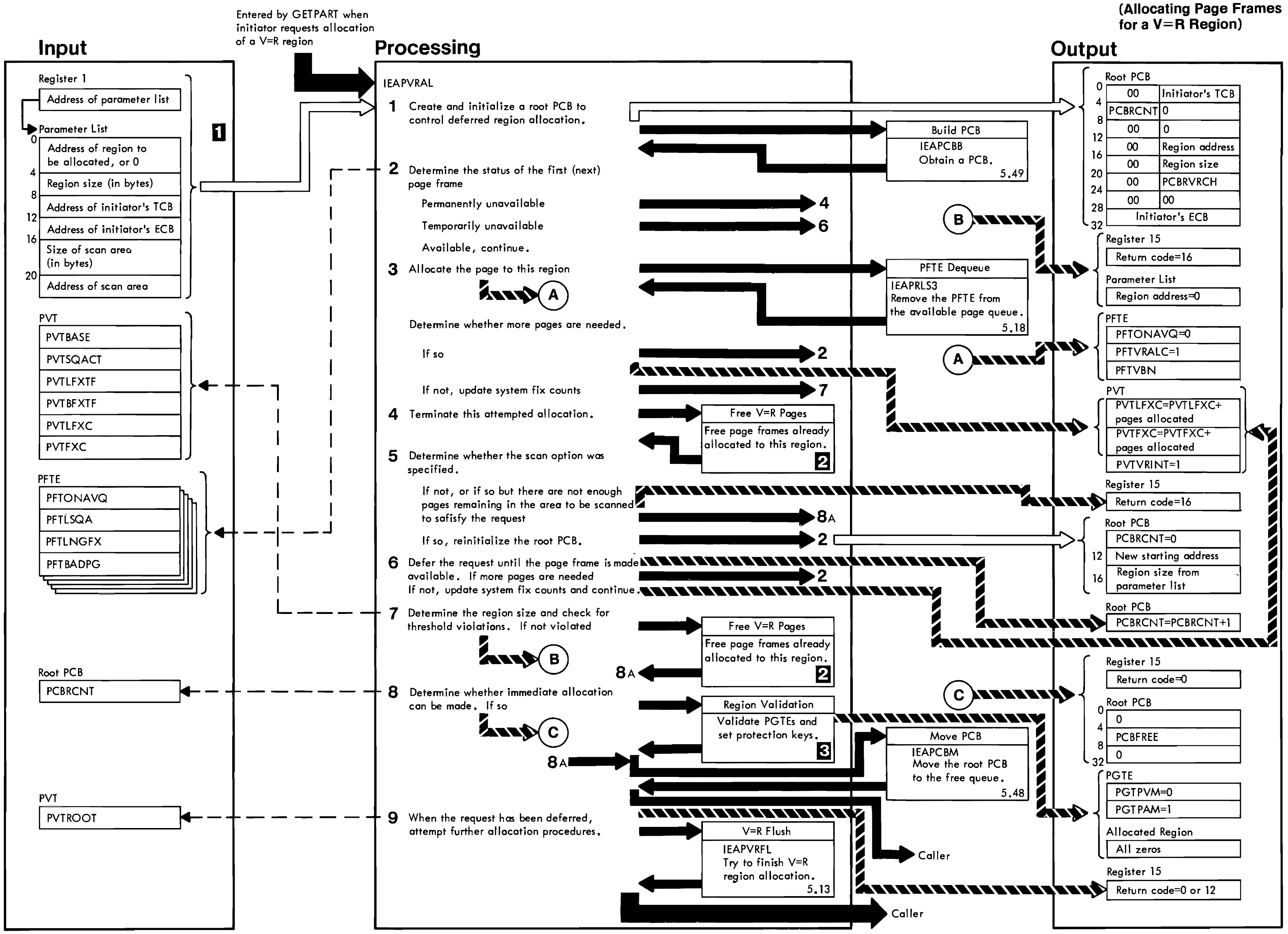
NOTES	ROUTINE NAME	LABEL
1 This is attempted when no pages exist on the available page queue or the number of available pages is less than the low threshold, and replacement is allowed (PVTRPLSW≠X'FF').	IEAPRSRL	AREPLNSH
2 This is done when allocation is suppressed and the reserve replenish queue is full.		ABYPASS
3 This is attempted when the reserve replenish queue is not full and there are pages available. IEAPMVPG is only called when the page is below the V=R line.		AAPCCHK
Note: See Flowchart 5-1 following notes for this diagram for processing logic.		

• Chart 5-1. Reserve Replenish Queue Processor



A2
APCCHECK
A1
IEAPRSRL
FROM:
QUEUE SCANNER
A2
PAGE AVAILABLE
YES
NO
SQARP
A3
RESERVE REPLENISH PAGES NEEDED
NO
YES
APCLTHR
A4
AVAILABLE PAGES NEEDED
NO
YES
F2
B2
B2
REACK
B2
REPLACEMENT ALLOWED
NO
YES
IEAP3
B3
IEAPRLS3
PFTEDQ REMOVE PAGE FROM AVAIL PAGE QUEUE
REPSCH
C2
IEAPRLS
FIND A REALLOCATABLE PAGE
C3
PAGE ABOVE THE V=R LINE
YES
NO
NOREPL
D2
RESERVE REPLENISH PAGES NEEDED
YES
NO
D1
TASK POST QUEUE PROCESSING NEEDED
NO
YES
H3
MOVE
D3
IEAPMVPG
GET A PAGE ABOVE THE V=R LINE
E1
INDICATE IEAPIOP SHOULD UNLOCK RESERVE REPLENISH QUEUE
SAPCTS
E2
PAGES AVAILABLE
NO
YES
SQANEED
E3
IEAPRLS2
PFTENQ PUT A PAGE ON RESERVE REPLENISH QUEUE
F2
ALLQ
F2
ALLOCATION PREVENTED
NO
YES
F3
DECREMENT THE COUNT OF RESERVE REPLENISH PAGES NEEDED
A2
RQS
G2
IEAPCRQS
RELEASE THE ALLOCATION QUEUE
H3
SUPPR
H2
SUPPRESS THE RESERVE REPLENISH QUEUE
NPS
H3
QUEUE SCANNER

Diagram 5.10
V=R Allocation Routine (Allocating Page Frames for a V=R Region)


Entered by GETPART when initiator requests allocation of a V=R region
Input
Register 1
Address of parameter list
Parameter List
0 Address of region to be allocated, or 0
4 Region size (in bytes)
8 Address of initiator's TCB
12 Address of initiator's ECB
16 Size of scan area (in bytes)
20 Address of scan area
1
PVT
PVTBASE
PVTSQACT
PVTLFXTF
PVTBFXTF
PVTLFXC
PVTFXC
PFTE
PFTONAVQ
PFTLSQA
PFTLNGFX
PFTBADPG
Root PCB
PCBRCNT
PVT
PVTROOT
Processing
IEAPVRAL
1 Create and initialize a root PCB to control deferred region allocation.
2 Determine the status of the first (next) page frame
Permanently unavailable 4
Temporarily unavailable 6
Available, continue.
3 Allocate the page to this region
A
Determine whether more pages are needed.
If so 2
If not, update system fix counts 7
4 Terminate this attempted allocation.
5 Determine whether the scan option was specified.
If not, or if so but there are not enough pages remaining in the area to be scanned to satisfy the request 8A
If so, reinitialize the root PCB. 2
6 Defer the request until the page frame is made available. If more pages are needed 2
If not, update system fix counts and continue.
7 Determine the region size and check for threshold violations. If not violated
B
8A
8 Determine whether immediate allocation can be made. If so
C
8A
9 When the request has been deferred, attempt further allocation procedures.
Build PCB
IEAPCBB
Obtain a PCB.
5.49
PFTE Dequeue
IEAPRLS3
Remove the PFTE from the available page queue.
5.18
Free V=R Pages
Free page frames already allocated to this region.
2
Free V=R Pages
Free page frames already allocated to this region.
2
Region Validation
Validate PGTEs and set protection keys.
3
Move PCB
IEAPCBM
Move the root PCB to the free queue.
5.48
V=R Flush
IEAPVRFL
Try to finish V=R region allocation.
5.13
Caller
Caller
B
A
C
Output
Root PCB
0 00 | Initiator's TCB
4 PCBRCNT | 0
8 00 | 0
12 00 | Region address
16 00 | Region size
20 00 | PCBRVRCH
24 00 | 00
28 Initiator's ECB
32
Register 15
Return code=16
Parameter List
Region address=0
PFTE
PFTONAVQ=0
PFTVRALC=1
PFTVBN
PVT
PVTLFXC=PVTLFXC+ pages allocated
PVTFXC=PVTFXC+ pages allocated
PVTVRINT=1
Register 15
Return code=16
Root PCB
PCBRCNT=0
12 New starting address
16 Region size from parameter list
Root PCB
PCBRCNT=PCBRCNT+1
Register 15
Return code=0
Root PCB
0 0
4 PCBFREE
8 0
32
PGTE
PGTPVM=0
PGTPAM=1
Allocated Region
All zeros
Register 15
Return code=0 or 12

• Diagram 5.10 V=R Allocation Routine (Module IEAPVEQR) (Module IEAPVEQR)

NOTES	ROUTINE NAME	LABEL
1 PVTPCVRA is turned on at entry and off at exit. Deferred allocation will be necessary if all needed pages are not immediately available.	IEAPVRAL	IEAPVRAL
2 The page frame is permanently unavailable if it is: • Allocated to SQA/LSQA (PFTLSQA=1) (unless TSO). • Long-fixed (PFTLNGFX=1). • Marked damaged after a machine check (PFTBADPG=1). The page is temporarily unavailable if it is currently in use for other than the above reasons.		STATUSCK
3 Page replacement is blocked during the call to the PFTE Dequeue routine (PVTBIGM=1 while PFTE Dequeue has control). An internal counter of page frames needed is decremented and tested. If it equals zero, no more pages are needed.		PROCPFTE
4 The length of the area examined = the beginning address of the region - (the beginning address of the unavailable page + 4096).		RET30
5 If the scan area size minus the unusable area size is greater than or equal to the requested size, the request can still be fulfilled.		RET30
6 The intercept bit is turned on (PVTVRINT=1). Those routines that make pages available test PVTVRINT prior to placing the PFTE on the available page queue. If PVTVRINT=1, they call IEAPVRS (Diagram 5.8-2) to allocate the page frame for a deferred request. For each fixed page, an internal counter is incremented to be used in threshold checking.		VRINT
7 The PFTE address is calculated as follows: PFTE address = $\frac{\text{Region address}}{256}$ +PFTE° address. PFTE° address is the address of the PFTE for the page frame whose real address is zero. The number of page frames to be allocated is equal to the region size divided by 4096. This is also the number of PFTEs that are examined to allocate this region. The new long-fix count that would result from this number of pages becoming allocated to V=R is calculated as follows: "count" = PVTBASE-PVTSQACT-PVTLXFTF-pages in request. If "count" is less than PVTLFXC, the threshold will be violated. The new fix count that would result from this number of pages becoming allocated to V=R is calculated as follows: "count" = PVTBASE-PVTSQACT-PVTBFXTF-pages in request + count of fixed page in this region. If "count" is less than PVTFXC, the threshold will be violated.		CALCPFTE

NOTES	ROUTINE NAME	LABEL
8 Immediate allocation can be made if all needed page frames were available (PCBRCNT=0).		RET10
9 The root PCB is placed on the V=R root PCB Queue and the V=R Flush routine is called to try to find more usable pages. Upon return, a test is made to determine whether the root PCB is still on the queue. If it is not, the request has been satisfied and the return code is set to zero. If it is still on the queue, the return code is set to 12 to indicate that the request has been deferred.		RET40
1 If the region address in the parameter list is zero, the Scan option has been selected and the address of the area to scan is used as the region address.		
2 The Free V=R Pages calculates the first PFTE in the region (region address/256 + PFTE° address) and handles each PFTE in the region beginning with that one as follows: • If the page is marked damaged by RMS (PFTBADPG=1), nothing is done to it. • If the page is not yet allocated to the region (PFTVRALC=0), PFTVRINT is set to zero. • If the page is not damaged and is allocated, the PFTE Enqueue routine (IEAPRLS2) is called to add the PFTE to the available page queue, PFTONAVQ is set to 1, and PFTVRALC is zeroed. When all pages in the region have been handled, return is made to the caller.	Free V=R Pages	
3 After the number of pages in the region is calculated, the Find Page routine (IEAPFP) is called to find the PGTE for the first (next) page. If the Find Page routine cannot find it, a call is made to IEAPSER to log the error (code 0302) and the next PFTE is processed. If the Find Page routine succeeds and the call is for allocation, the PGTE is validated by setting PGTPVM to 0, PGTPAM to 1, and the protection key to 0. The next PFTE is accessed and the call to the Find Page routine is repeated. When all pages have been validated, the entire region is cleared to zeros, and control is returned to the caller. If the Find Page routine succeeds and the call is for region freeing, the PGTE is invalidated by setting PGTPVM to 1 and PGTPAM to 0. The next PFTE is accessed, and the call to the Find Page routine is repeated. When all pages have been invalidated, control is returned to the caller.	Region Validation	

Diagram 5.11
V=R Release Routine
(Allocating an Intercepted Page to a V=R Region)

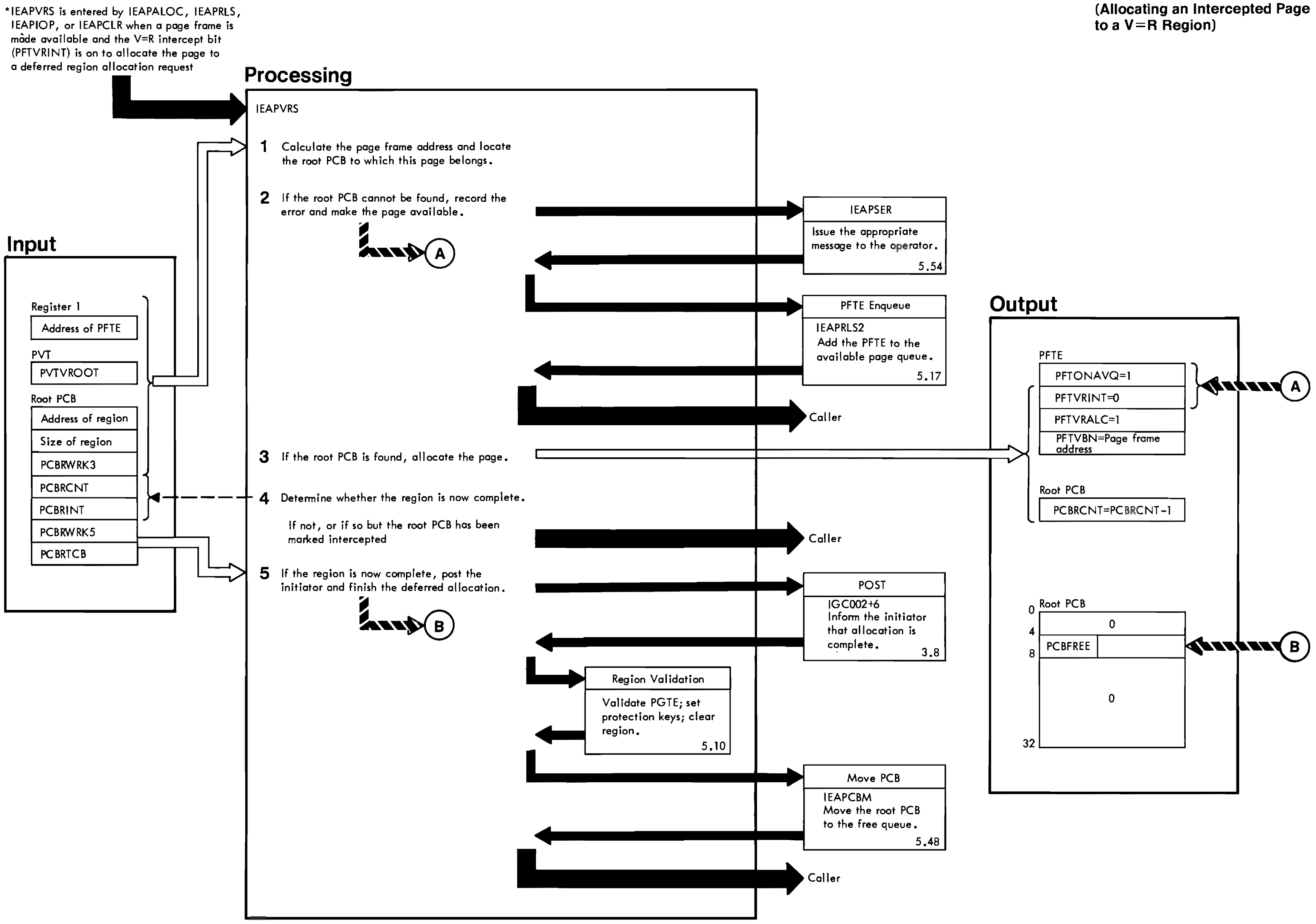
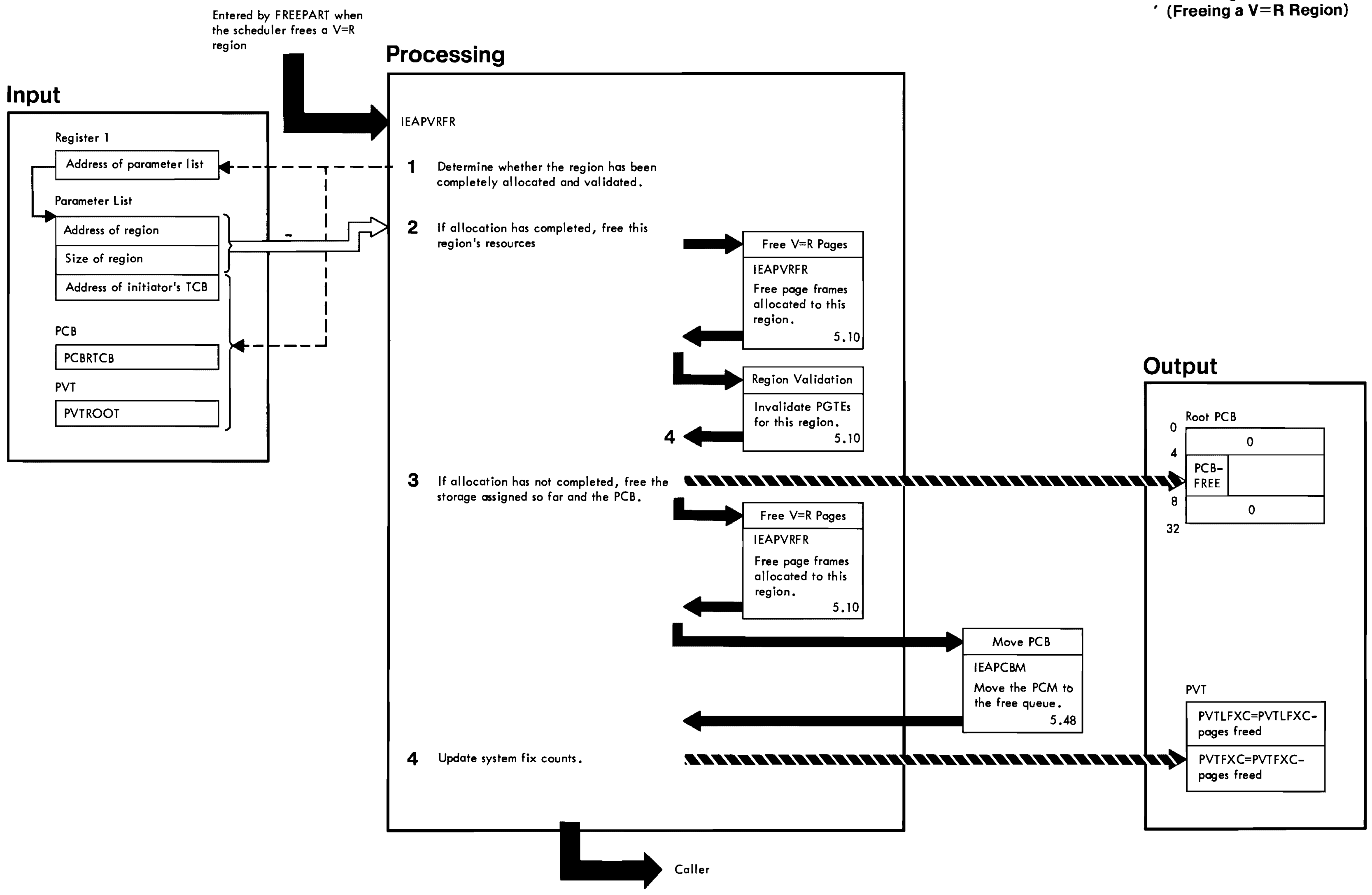

*IEAPVRS is entered by IEAPALOC, IEAPRLS, IEAPIOP, or IEAPCLR when a page frame is made available and the V=R intercept bit (PFTVRINT) is on to allocate the page to a deferred region allocation request
Input
Register 1
Address of PFTE
PVT
PVTVROOT
Root PCB
Address of region
Size of region
PCBRWRK3
PCBRCNT
PCBRINT
PCBRWRK5
PCBRTCB
Processing
IEAPVRS
1 Calculate the page frame address and locate the root PCB to which this page belongs.
2 If the root PCB cannot be found, record the error and make the page available.
A
3 If the root PCB is found, allocate the page.
4 Determine whether the region is now complete.
If not, or if so but the root PCB has been marked intercepted
5 If the region is now complete, post the initiator and finish the deferred allocation.
B
IEAPSER
Issue the appropriate message to the operator.
5.54
PFTE Enqueue
IEAPRLS2
Add the PFTE to the available page queue.
5.17
Caller
Caller
POST
IGC002+6
Inform the initiator that allocation is complete.
3.8
Region Validation
Validate PGTE; set protection keys; clear region.
5.10
Move PCB
IEAPCBM
Move the root PCB to the free queue.
5.48
Caller
Output
PFTE
PFTONAVQ=1
PFTVRINT=0
PFTVRALC=1
PFTVBN=Page frame address
A
Root PCB
PCBRCNT=PCBRCNT-1
Root PCB
0
4
8
32
0
PCBFREE
0
B

Diagram 5.11 V=R Release Routine (Module IEAPVEQR)

NOTES		ROUTINE NAME	LABEL
1	The Root PCB (on the queue headed by PVTVROOT) is located by comparing the page frame address to the region and region end address. (Page frame address = ((PFTE index)*256)+PFTE address.) If the page frame address is greater than or equal to the region address and less than the region end address (region address + region size), the root PCB has been located.	IEAPVRS	VAREL10
2	IEAPSER is called with a 0301 error code. PFTONAVQ is set to 1 and PFTVRINT is set to 0 to indicate that the page is no longer being used and that no outstanding requests exist for it.		PVTVSER
3	The count of pages needed to complete the region (PCBRCNT) is decremented.		VRREL10
4	If PCBRCNT=0, the region is now complete. However, if interception is indicated (PCERINT=1), the ECB has already been posted (code 16).		VRREL50
5			

Diagram 5.12
V=R Region Free Routine
(Freeing a V=R Region)



Entered by FREEPART when the scheduler frees a V=R region
Input
Register 1
Address of parameter list
Parameter List
Address of region
Size of region
Address of initiator's TCB
PCB
PCBRTCB
PVT
PVTROOT
Processing
IEAPVRFR
1 Determine whether the region has been completely allocated and validated.
2 If allocation has completed, free this region's resources
Free V=R Pages
IEAPVRFR
Free page frames allocated to this region.
5.10
Region Validation
Invalidate PGTEs for this region.
5.10
4
3 If allocation has not completed, free the storage assigned so far and the PCB.
Free V=R Pages
IEAPVRFR
Free page frames allocated to this region.
5.10
Move PCB
IEAPCBM
Move the PCM to the free queue.
5.48
4 Update system fix counts.
Caller
Output
Root PCB
0
4
8
32
0
PCB-FREE
0
PVT
PVTLFXC=PVTLFXC-pages freed
PVTFXC=PVTFXC-pages freed

•Diagram 5.12 V=R Region Free Routine (Module IEAPVEQR)

NOTES	ROUTINE NAME	LABEL
1 PVTPCVRF is turned on at entry and off at exit. The V=R Root Queue (headed by PVTVROOT) is searched for a root PCB whose TCB pointer (PCBRTCB) matches the TCB address in the parameter list. If no match is found, the region has previously been completed and validated.	IEAPVRFR	VRFREE10
2 The region address and size (from the parameter list) are placed in an an internal PCB and passed to the Free V=R Pages routine.		VRFREE80
4 The number of pages freed (region size divided by 4096) is subtracted from the fix counts.		

Diagram 5.13
V=R Flush Routine
(Attempting to Get Needed V=R Page Frames)

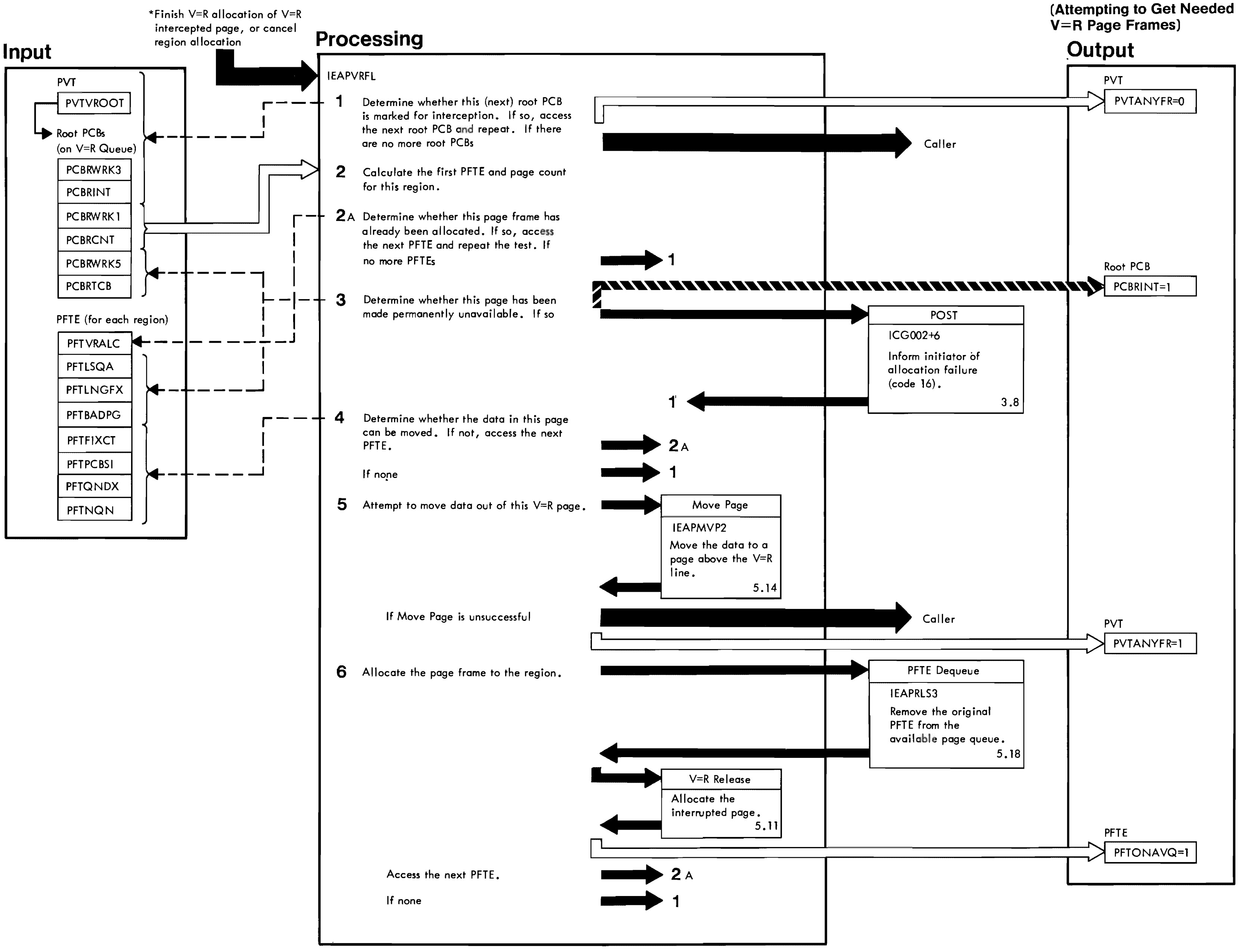
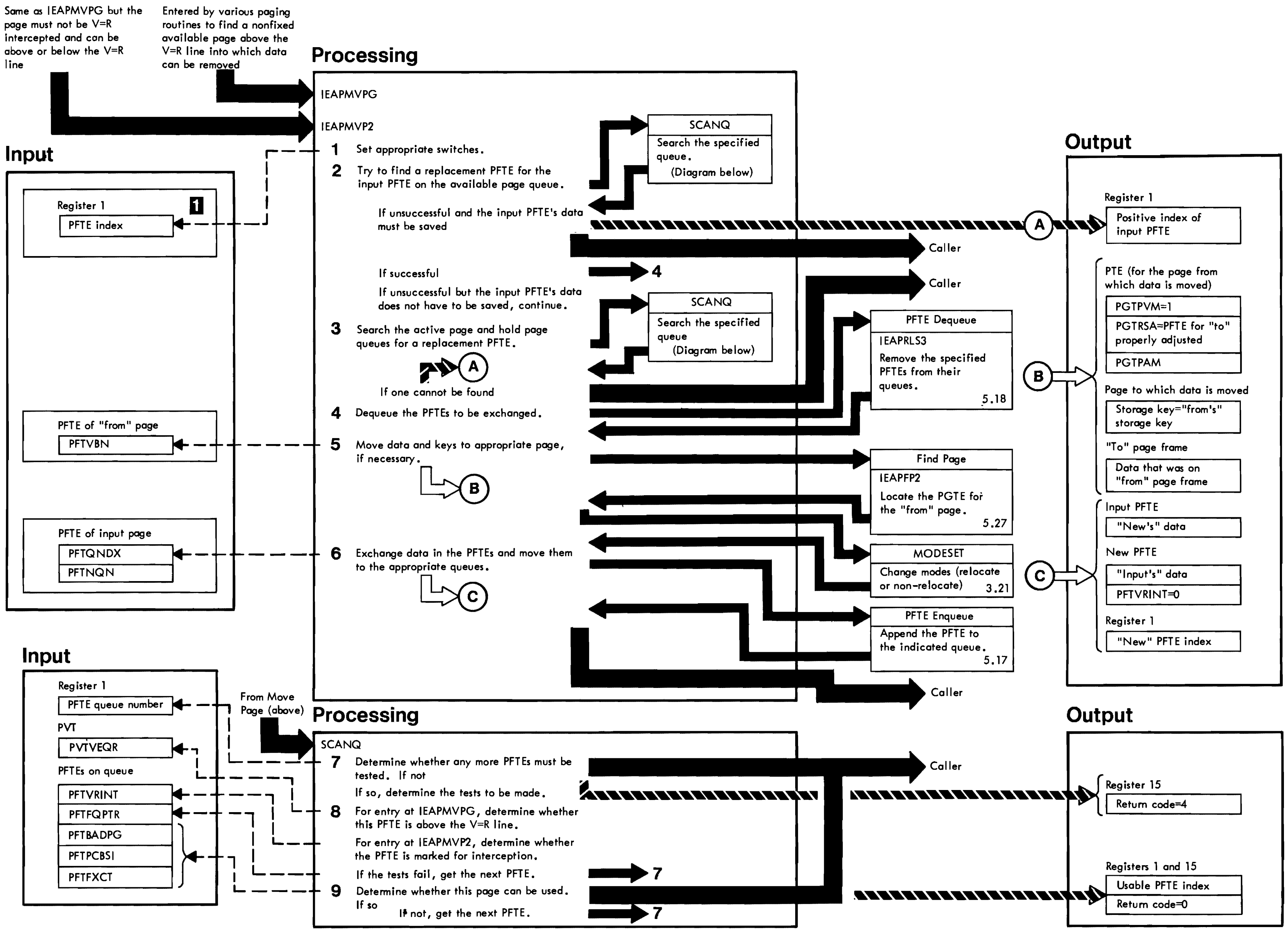

Input
PVT
PVTVROOT
Root PCBs
(on V=R Queue)
PCBRWRK3
PCBRINT
PCBRWRK1
PCBRCNT
PCBRWRK5
PCBRTCB
PFTE (for each region)
PFTVRALC
PFTLSQA
PFTLNGFX
PFTBADPG
PFTFIXCT
PFTPCBSI
PFTQNDX
PFTNQN
*Finish V=R allocation of V=R intercepted page, or cancel region allocation
Processing
IEAPVRFL
1 Determine whether this (next) root PCB is marked for interception. If so, access the next root PCB and repeat. If there are no more root PCBs
Caller
2 Calculate the first PFTE and page count for this region.
2A Determine whether this page frame has already been allocated. If so, access the next PFTE and repeat the test. If no more PFTEs
1
3 Determine whether this page has been made permanently unavailable. If so
POST
ICG002+6
Inform initiator of allocation failure (code 16).
3.8
1
4 Determine whether the data in this page can be moved. If not, access the next PFTE.
2A
If none
1
5 Attempt to move data out of this V=R page.
Move Page
IEAPMVP2
Move the data to a page above the V=R line.
5.14
If Move Page is unsuccessful
Caller
6 Allocate the page frame to the region.
PFTE Dequeue
IEAPRLS3
Remove the original PFTE from the available page queue.
5.18
V=R Release
Allocate the interrupted page.
5.11
Access the next PFTE.
2A
If none
1
Output
PVT
PVTANYFR=0
Root PCB
PCBRINT=1
PVT
PVTANYFR=1
PFTE
PFTONAVQ=1

Diagram 5.13 V=R Flush Routine (Module IEAPVEQR)

NOTES	ROUTINE NAME	LABEL
* Entered by: • IEAPVRAL when immediate V=R region allocation cannot be completed. • Free subroutine when a V=R intercepted page becomes free. IEAPIOP when a page-in completes for a V=R intercepted page.	IEAPVRFL	
1 The root PCBs tested are on the V=R queue pointed to by PVTVROOT and chained by the PCBRWRK3 fields.		VRFLSH20
2 The first PFTE address is calculated by accessing the region address (PCBRWRK3), generating a 16 bit PFTE index, and adding to that the apparent page frame table origin.		VRFLSH30
3 The page is permanently unavailable if it is: Allocated to SQA or LSQA (PFTLSQA=1) (unless allocated to a TSO task). Long-fixed (PFTLNGFX=1). Marked damaged by RMS (PFTBADPG=1). If any pages are permanently unavailable, the root PCB is marked for interception.		VRFLSH40
4 The page cannot be moved if it is: • Fixed (PFTFIXCT≠0). • Involved in any paging supervisor activity (PFTPCBSI=1 or PFTQNDX=PFTNQN).		VRFLSH45
5 If the Move Page routine fails, there are no more pages available queue and none could be freed so allocation allocation cannot complete for this or any other region.		VRFLSH70
6 Replacement is prohibited before allocation and allowed again afterward (PVTBIGM is set to 1 before and 0 after.).		VRFLSH70



Same as IEAPMVPG but the page must not be V=R intercepted and can be above or below the V=R line

Entered by various paging routines to find a nonfixed available page above the V=R line into which data can be removed

Processing

IEAPMVPG

IEAPMVP2

1 Set appropriate switches.

2 Try to find a replacement PFTE for the input PFTE on the available page queue.

If unsuccessful and the input PFTE's data must be saved

If successful

If unsuccessful but the input PFTE's data does not have to be saved, continue.

3 Search the active page and hold page queues for a replacement PFTE.

If one cannot be found

4 Dequeue the PFTEs to be exchanged.

5 Move data and keys to appropriate page, if necessary.

6 Exchange data in the PFTEs and move them to the appropriate queues.

SCANQ — Search the specified queue. (Diagram below)

SCANQ — Search the specified queue (Diagram below)

PFTE Dequeue — IEAPRLS3 — Remove the specified PFTEs from their queues. 5.18

Find Page — IEAPFP2 — Locate the PGTE for the "from" page. 5.27

MODESET — Change modes (relocate or non-relocate) 3.21

PFTE Enqueue — Append the PFTE to the indicated queue. 5.17

Input

Register 1 — PFTE index

PFTE of "from" page — PFTVBN

PFTE of input page — PFTQNDX, PFTNQN

Output

Register 1 — Positive index of input PFTE

PTE (for the page from which data is moved) — PGTPVM=1; PGTRSA=PFTE for "to" properly adjusted; PGTPAM

Page to which data is moved — Storage key="from's" storage key

"To" page frame — Data that was on "from" page frame

Input PFTE — "New's" data

New PFTE — "Input's" data; PFTVRINT=0

Register 1 — "New" PFTE index

Input

Register 1 — PFTE queue number

PVT — PVTVEQR

PFTEs on queue — PFTVRINT, PFTFQPTR, PFTBADPG, PFTPCBSI, PFTFXCT

From Move Page (above)

Processing

SCANQ

7 Determine whether any more PFTEs must be tested. If not

If so, determine the tests to be made.

8 For entry at IEAPMVPG, determine whether this PFTE is above the V=R line.

For entry at IEAPMVP2, determine whether the PFTE is marked for interception.

If the tests fail, get the next PFTE.

9 Determine whether this page can be used. If so

If not, get the next PFTE.

Output

Register 15 — Return code=4

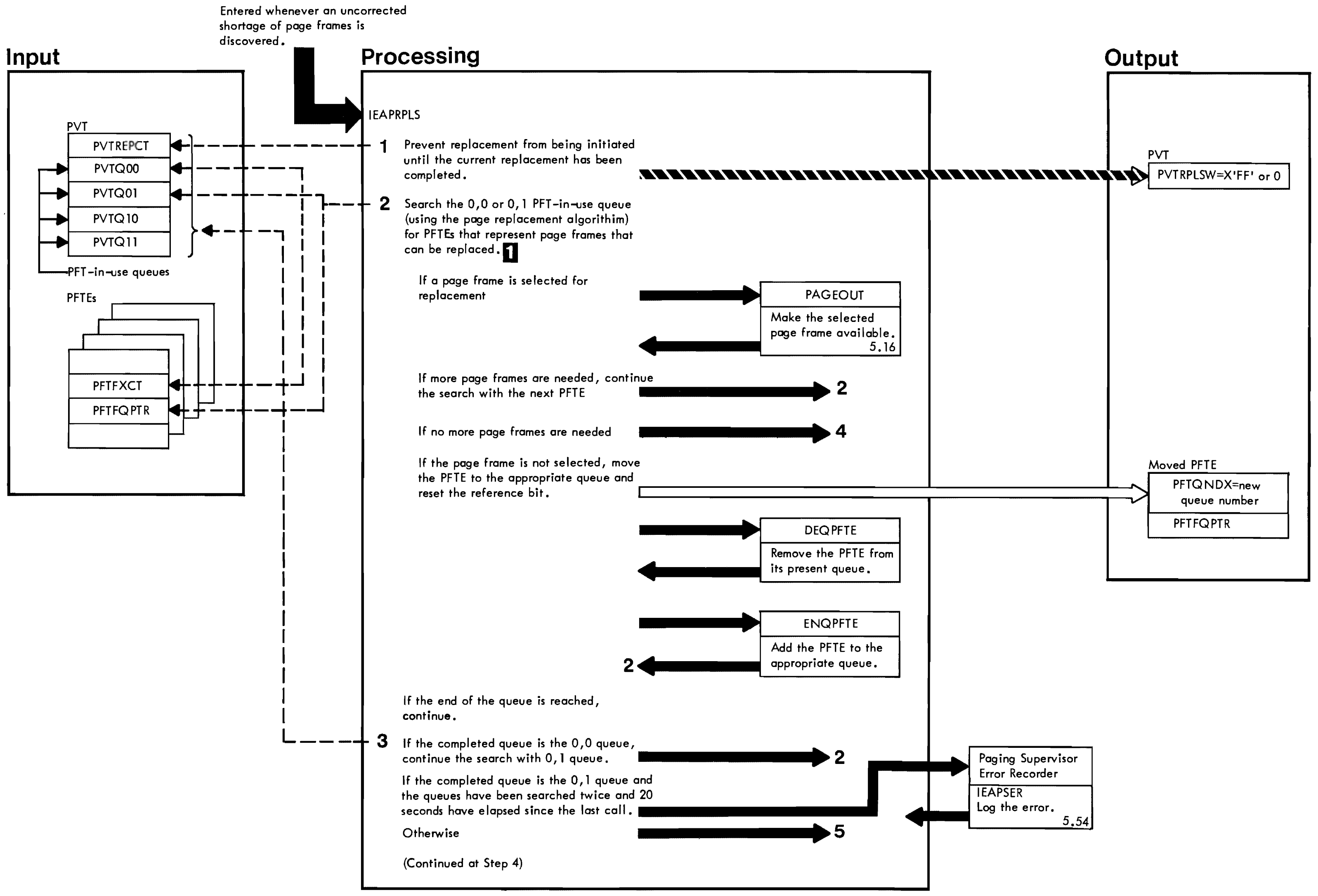
Registers 1 and 15 — Usable PFTE index; Return code=0

Diagram 5.14 Move Page Routine (Module IEAPVEQR)

NOTES	ROUTINE NAME	LABEL
1 If register 1 contains a positive PFTE index, the data and storage keys of the input page frame must be preserved in any move. If register 1 contains a negative PFTE index, the data and keys of the input page frame do not have to be saved.	IEAPMVPG	
1 An internal entry switch is set to 0 for entry at IEAPMVPG; to 1 for entry at IEAPMVP2. An internal "I" switch is set to 1 if Register 1 is positive; to 0 if Register 1 is negative.	IEAPMVPG IEAPMVP2	ISWTEST
2 If a PFTE is not found on the available page queue and: • The "I" switch = 1, a "one-way" move cannot be made. • The "I" switch = 0, the data and keys in the input page frame need not be saved, but those in the "new" page frame (the one into which data will be moved) must be saved. An internal "M" switch is set to indicate this. If a PFTE is found on the available page queue, and the "I" switch equals zero, the "M" switch is set to 0.		ICONT
3 SCANQ is called to scan each active page queue and then the hold page queue until a usable PFTE is found.		SCANACT
4 Page replacement is prevented by setting PVTBIGM before calling the PFTE Dequeue routine. The PFTE Dequeue routine is then called twice; first for the input PFTE, then for the new PFTE.		DEV

NOTES	ROUTINE NAME	LABEL
5 If the "I" switch is on, the data from the input page is moved into the new page frame. If the "M" switch is on, the data from the new page is moved into the input page frame. If neither switch is on, no data is moved. Upon return from the Find Page routine, a PTLB instruction is issued to purge the hardware DLATs after invalidating the PTE for the page from which data was moved. MODESET is called before the move to place the routine in nonrelocation mode and after the move to return it to relocation mode.		IORMCOM
6 If the input PFTE was not on a queue originally (PFTQNDX=PFTNQN), the first call to the PFTE Enqueue routine is skipped. Otherwise, PFTE Enqueue is called twice; once to add the new PFTE to the queue of the input PFTE and once to add the input PFTE to the new PFTE's queue. Replacement is allowed (PVTBIGM=0) before exit is taken.		NONE
7 A register loaded with PFTQPTR is tested for zero to determine whether any PFTEs are left to be tested. The entry switch (set in IEAPMVPG or IEAPMVP2) is tested to determine the entry point.	SCANQ	SCAN
8 The entry switch (set in IEAPMVPG or IEAPMVP2) is tested to determine the entry point.		TSET and TESTINT
9 The page is usable if it is: • Not marked as damaged by RMS (PFTBADPG=0). • Not assigned a PCB (PFTPCBSI=0). • Not fixed (PFTFXCT=0).		BAD

• Diagram 5.15 (Steps 1-3) Page Replacement Routine



Entered whenever an uncorrected shortage of page frames is discovered.

IEAPRPLS

1 Prevent replacement from being initiated until the current replacement has been completed.

2 Search the 0,0 or 0,1 PFT-in-use queue (using the page replacement algorithim) for PFTEs that represent page frames that can be replaced. 1

If a page frame is selected for replacement

PAGEOUT: Make the selected page frame available. 5.16

If more page frames are needed, continue the search with the next PFTE

If no more page frames are needed

If the page frame is not selected, move the PFTE to the appropriate queue and reset the reference bit.

DEQPFTE: Remove the PFTE from its present queue.

ENQPFTE: Add the PFTE to the appropriate queue.

If the end of the queue is reached, continue.

3 If the completed queue is the 0,0 queue, continue the search with 0,1 queue.

If the completed queue is the 0,1 queue and the queues have been searched twice and 20 seconds have elapsed since the last call.

Paging Supervisor Error Recorder — IEAPSER: Log the error. 5.54

Otherwise

(Continued at Step 4)

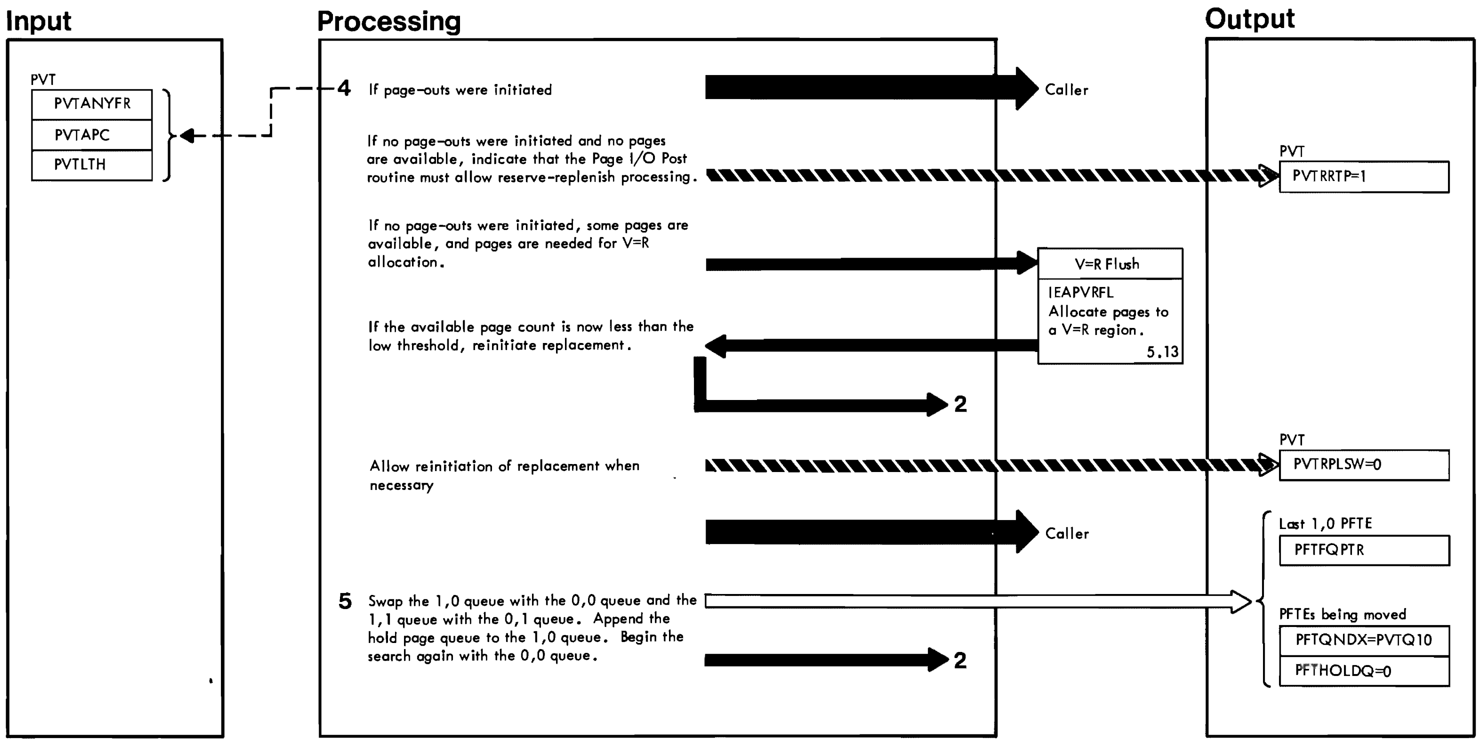
• Diagram 5.15 (Steps 1-3) Page Replacement Routine (Module IEAPRPLS)

NOTES	ROUTINE NAME	LABEL
1 This replacement attempt is not finished until all I/O it initiates is completed. An internal counter of pages needed is set to PVTREPCT and a counter of scan iterations (PVT00) is set to zero. The 0,0 PFTE queue is set as the first queue to be scanned.	IEAPRPLS	GORPLACE
2 The queue is located by taking the physical PFTE queue number from PVTQ00 or PVTQ01 and using that number as an index into the PFTE queue headers in the PVT. See the page replacement algorithm (on the following page) for an explanation of the scan logic. Upon return from PAGEOUT, the counter of pages needed is decremented by one. If it now equals zero, no more pages are needed.		RPLACE15
3 The internal counter of scan iterations is tested. If it is equal to 3, two iterations have been made through the PFTE queues. This is an error condition indicating either that replacement was invoked needlessly or that fixed, SQA, LSQA, V=R, or any combination of these types of pages have overcommitted real storage and left an insufficient number of pages for paging. The error code passed to IEAPSER is 0600. **1** See table following the notes for this diagram for an explanation of the page replacement algorithm.		ENOFQ10

PAGE REPLACEMENT ALGORITHM TABLE

Queue being Scanned	Fix count	Reference Bit setting	Change Bit setting	Action
0,0	non-0			Page not eligible for replacement; continue search with next PFTE.
0,0	0	0	0	Replace the page frame associated with this PFTE.
0,0	0	0	1	*Move this PFTE to the 0,1 queue and continue search with the next PFTE.
0,0	0	1	0	*Move this PFTE to the 1,0 queue and continue search with the next PFTE.
0,0	0	1	1	*Move this PFTE to the 1,1 queue and continue search with the next PFTE.
0,1	non-0			Page not eligible for replacement; continue search with next PFTE.
0,1	0	0	1	Replace the page frame associated with this PFTE.
0,1	0	1	1	*Move this PFTE to the 1,1 queue and continue search with the next PFTE.

* The Reference Bit is set to zero each time the PFTE representing the page frame is moved.



4 If page-outs were initiated

If no page-outs were initiated and no pages are available, indicate that the Page I/O Post routine must allow reserve-replenish processing.

If no page-outs were initiated, some pages are available, and pages are needed for V=R allocation.

If the available page count is now less than the low threshold, reinitiate replacement.

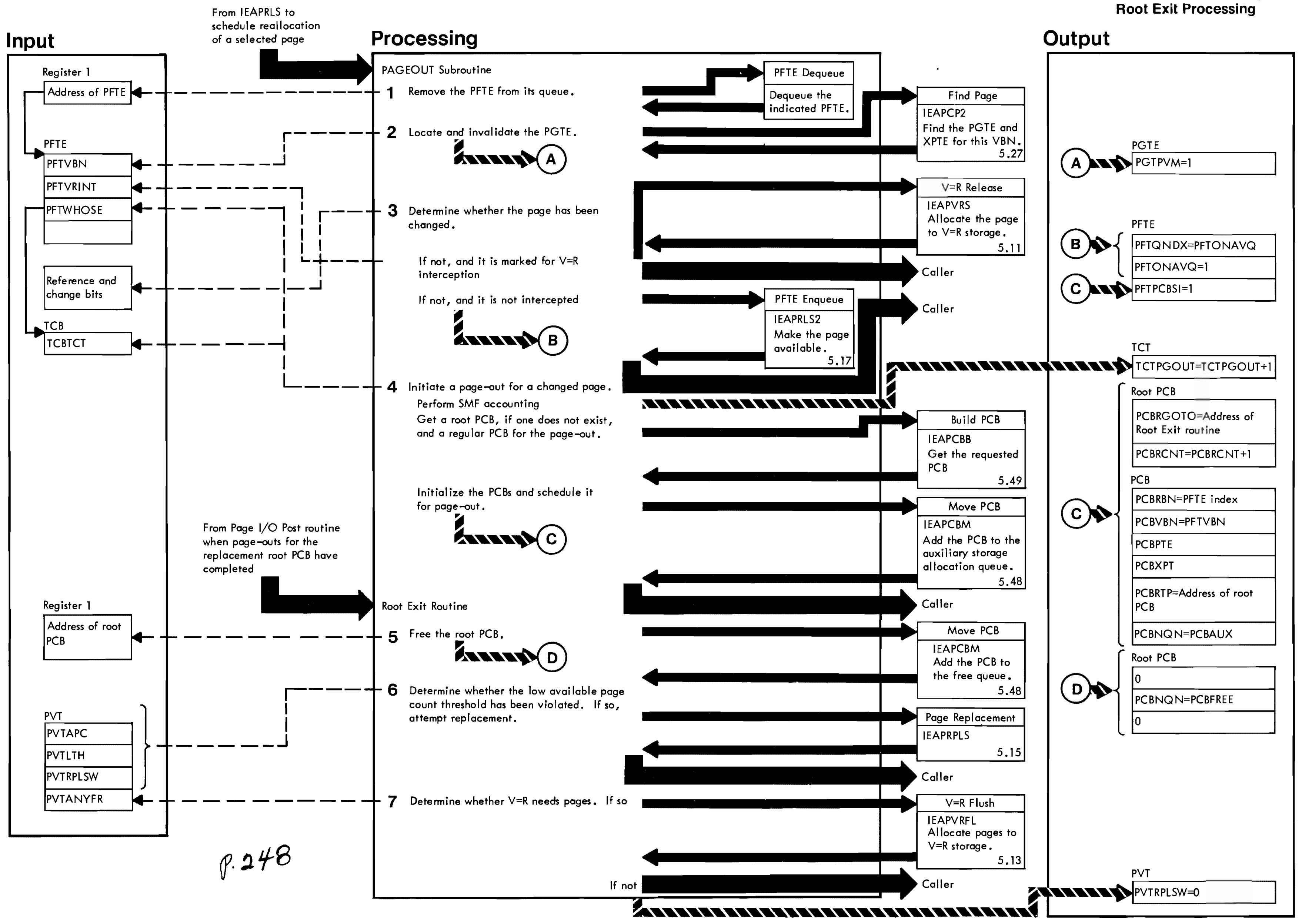
Allow reinitiation of replacement when necessary

5 Swap the 1,0 queue with the 0,0 queue and the 1,1 queue with the 0,1 queue. Append the hold page queue to the 1,0 queue. Begin the search again with the 0,0 queue.

Diagram 5.15 (Steps 4-5) Page Replacement Routine (Module IEAPRPLS)

NOTES	ROUTINE NAME	LABEL
4	IEAPRPLS	ENDPROC
5 The scan iteration counter is incremented before the swap. The queues are swapped by exchanging the physical queue numbers in PVTQ00 with PVTQ10 and PVTQ01 with PVTQ11. The hold page queue is appended by setting the forward pointer of the last PFTE on the 1,0 queue to the first PFTE on the hold page queue, turning off PFTHOLDQ in the appended PFTEs, and setting their PFTQNDX fields to the value in PVTQ10.		ENDOFQ20

Diagram 5.16
Page Replacement — Allocation Scheduling and Root Exit Processing



PAGEOUT Subroutine

1 Remove the PFTE from its queue.

2 Locate and invalidate the PGTE. (A)

3 Determine whether the page has been changed.

If not, and it is marked for V=R interception

If not, and it is not intercepted (B)

4 Initiate a page-out for a changed page.

Perform SMF accounting

Get a root PCB, if one does not exist, and a regular PCB for the page-out.

Initialize the PCBs and schedule it for page-out. (C)

Root Exit Routine

5 Free the root PCB. (D)

6 Determine whether the low available page count threshold has been violated. If so, attempt replacement.

7 Determine whether V=R needs pages. If so

p. 248

• Diagram 5.16 Page Replacement -- Allocation Scheduling and Root Exit Processing (Module IEAPRPLS)

NOTES	ROUTINE NAME	LABEL
1	PAGEOUT	PAGEOUT
2 The PGTE is invalidated since the page frame is being made unavailable. If the return code from FINDPAGE is not zero, PAGEOUT passes control to IEAPSER, with code 601, to issue a message indicating that the error is recoverable. Then processing continues at step 3, treating the page as not changed.		
3 If the page is unchanged, an exact copy of it exists on external page storage (that is, a page-out is not needed). If the page is given to V=R Release, it is not counted toward the replacement count.		POUT30
4 SMF accounting routines record the page-out operation in the effected task's statistics table (TCT) if one exists. If PFTWHOSE=0 or TCBTCT=0, there is no TCT.		POUT00
5 Since all operations represented by this root PCB have been completed, the root PCB is no longer needed.	Root Exit Routine	ROOTEXIT
6		RTEXIT01
7		RTEXIT02

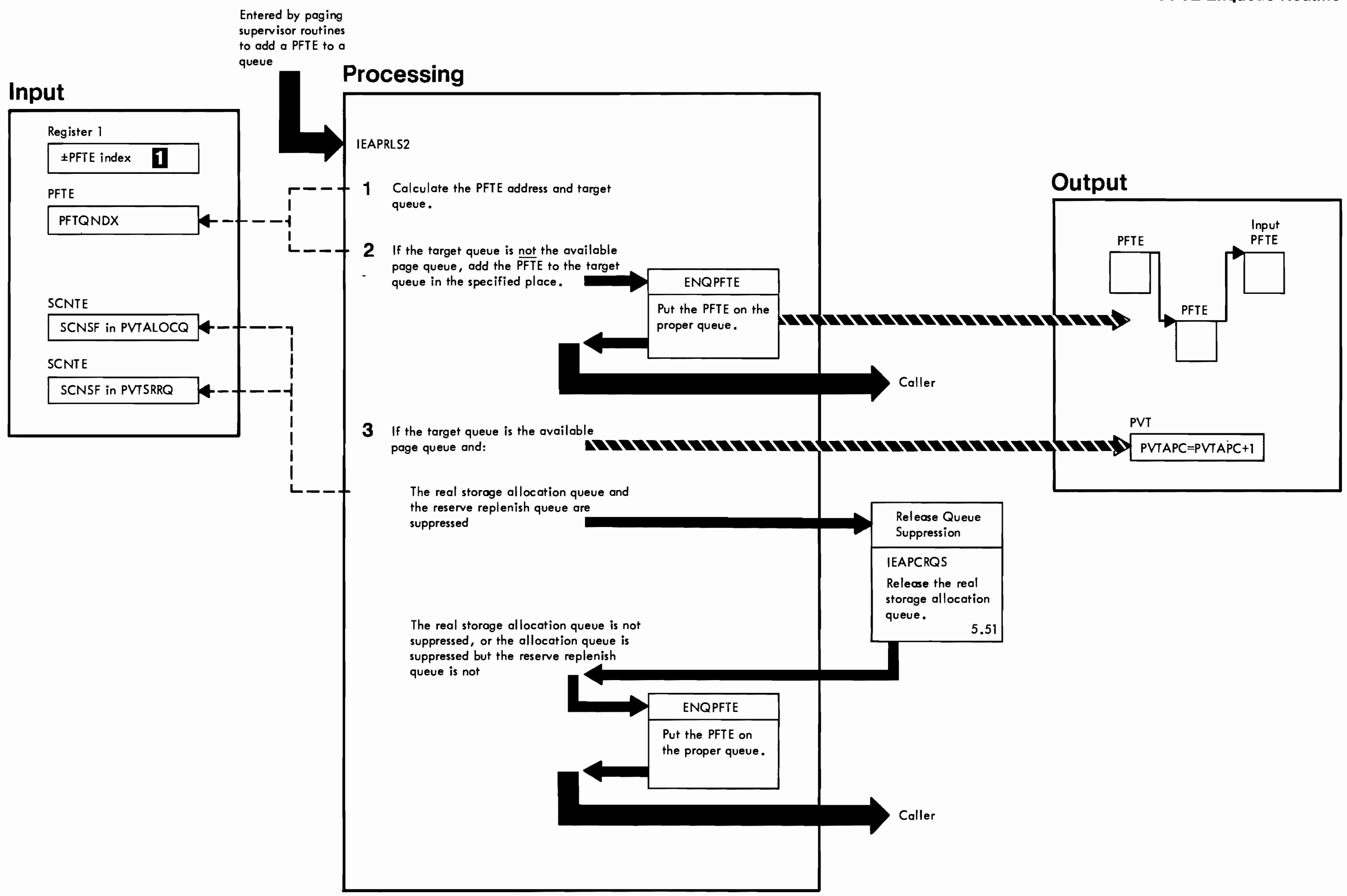
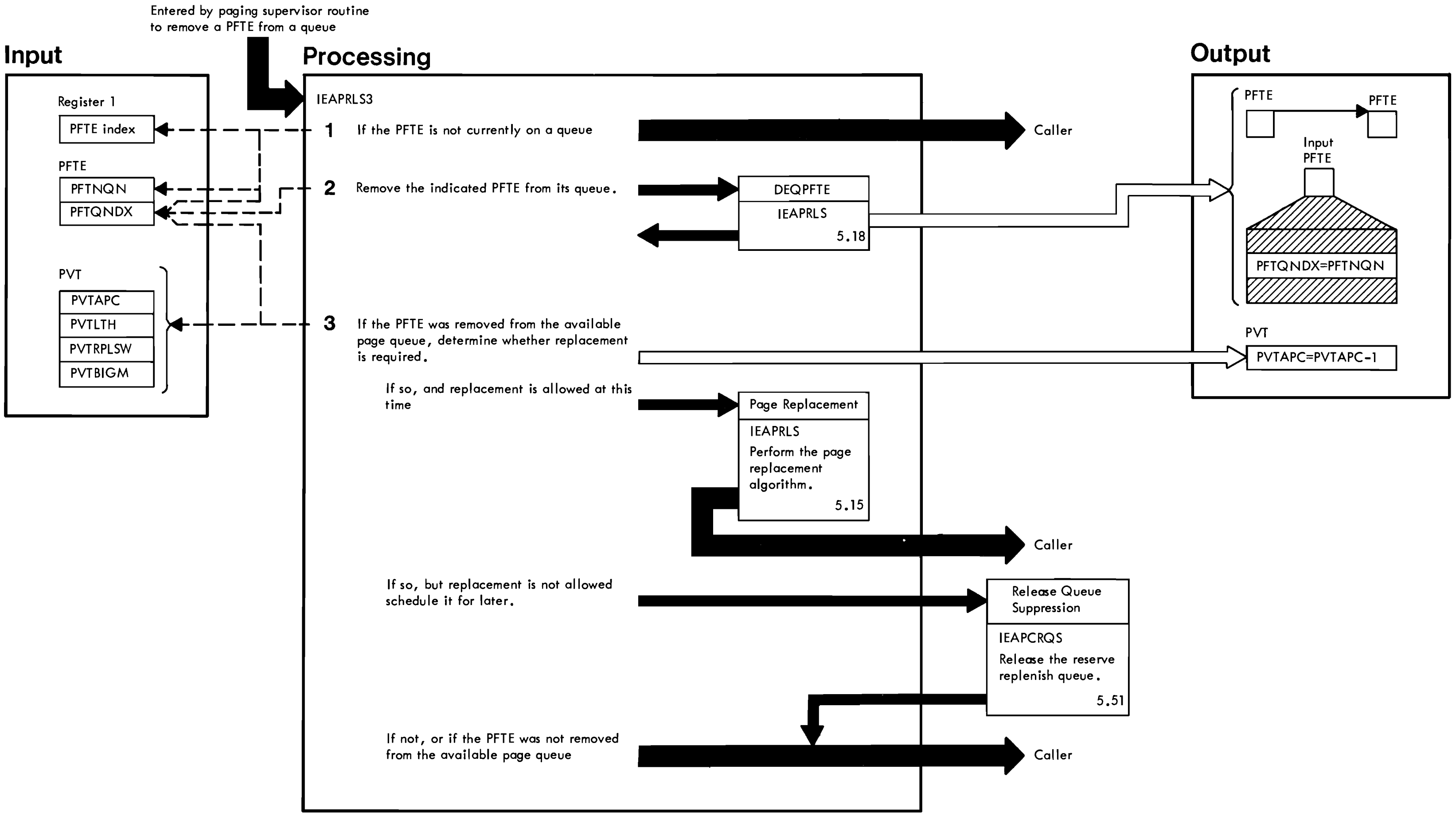

Entered by paging supervisor routines to add a PFTE to a queue
Input
Register 1
±PFTE index 1
PFTE
PFTQNDX
SCNTE
SCNSF in PVTALOCQ
SCNTE
SCNSF in PVTSRRQ
Processing
IEAPRLS2
1 Calculate the PFTE address and target queue.
2 If the target queue is not the available page queue, add the PFTE to the target queue in the specified place.
ENQPFTE
Put the PFTE on the proper queue.
Caller
3 If the target queue is the available page queue and:
The real storage allocation queue and the reserve replenish queue are suppressed
Release Queue Suppression
IEAPCRQS
Release the real storage allocation queue.
5.51
The real storage allocation queue is not suppressed, or the allocation queue is suppressed but the reserve replenish queue is not
ENQPFTE
Put the PFTE on the proper queue.
Caller
Output
PFTE
Input PFTE
PFTE
PVT
PVTAPC=PVTAPC+1

Diagram 5.17 PFTE Enqueue Routine (Module IEAPRPLS)

NOTES	ROUTINE NAME	LABEL
1 If register 1 is positive, the PFTE is to be added tc the bottom of the queue; if negative, the PFTE is added to the top.	IEAPRLS2	
1 The input PFTE index is complemented, if necessary, and converted to the PFTE address. The target queue is located by using PFTQNDX as an index into the PFT queue headers in the PVT.		IEAPRLS2
2 If PFTQNDX≠PFTAVQN, the target is not the available page queue.		NQ20
3 The available page count (PVTAPC) is incremented by 1 to indicate that another page is immediately available for system use.		NQ10

Diagram 5.18
PFTE Dequeue Routine



1 If the PFTE is not currently on a queue

2 Remove the indicated PFTE from its queue.

3 If the PFTE was removed from the available page queue, determine whether replacement is required.

If so, and replacement is allowed at this time

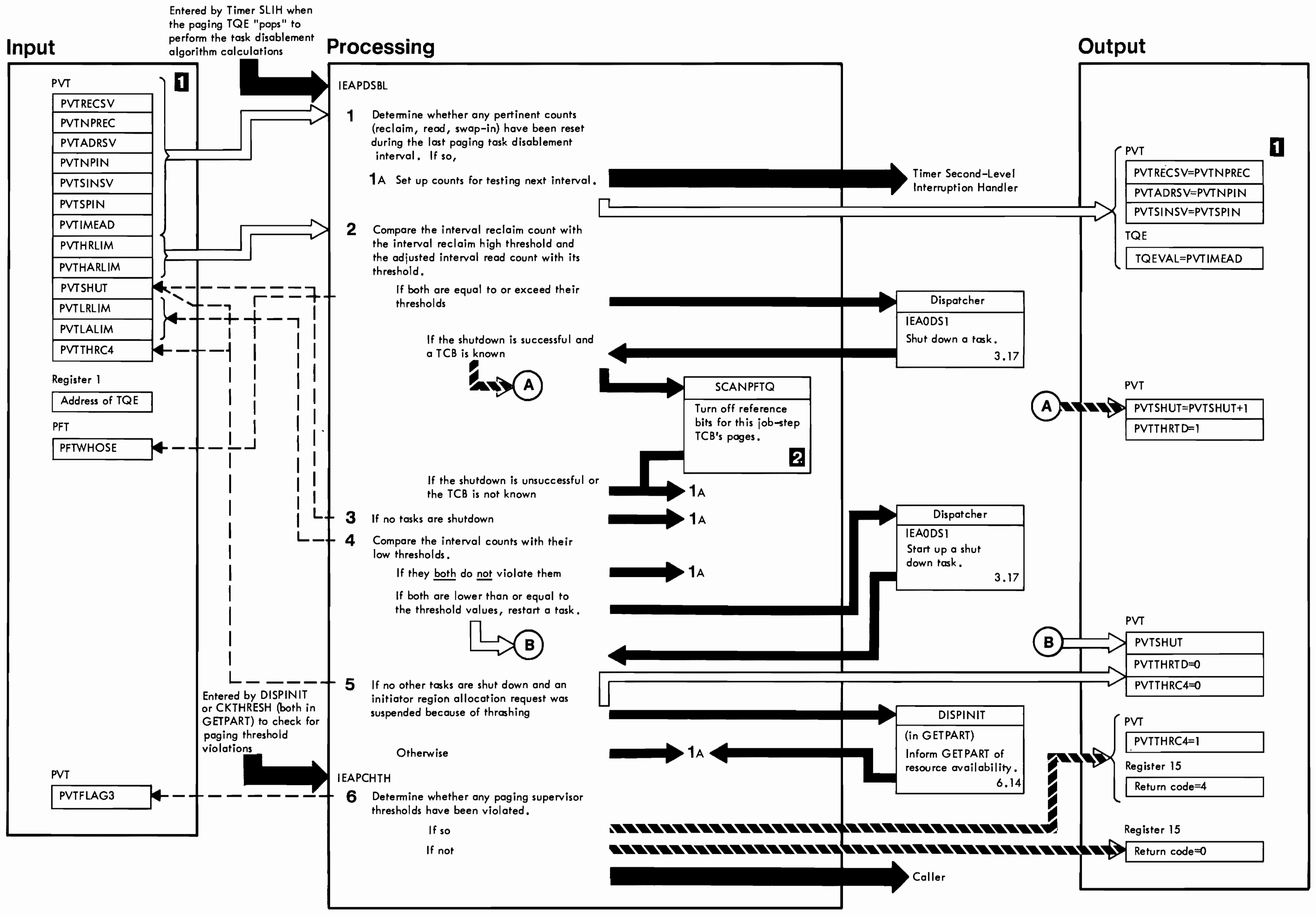
If so, but replacement is not allowed schedule it for later.

If not, or if the PFTE was not removed from the available page queue

Diagram 5.18 PFTE Dequeue Routine (Module IEAPRPLS)

NOTES	ROUTINE NAME	LABEL
1 The input PFTE index is converted to the PFTE address. The target queue is located by using PFTQNDX as an index into the PFT queue headers in the PVT.	IEAPRLS3	IEAPRLS3
3 The available page count (PVTAPC) is decremented by 1 and compared to the low threshold (PVTLTH). The Page Replacement routine is called immediately if: • PVTLTH>PVTAPC and • PVTRLPSW=0 (replacement is not in progress) and • PVTBIGM=0 (replacement is allowed). If PVTBIGM=1, replacement is not allowed and the available page count is tested again. If PVTAPC=0, replacement is scheduled for later by releasing the SQA reserve replenish queue. Otherwise, the call to the Release Queue Suppression routine (IEAPCRQS) is skipped.		

Diagram 5.19
Task Disablement Algorithm and Threshold Checking


Input
Entered by Timer SLIH when the paging TQE "pops" to perform the task disablement algorithm calculations
Processing
Output
PVT
1
PVTRECSV
PVTNPREC
PVTADRSV
PVTNPIN
PVTSINSV
PVTSPIN
PVTIMEAD
PVTHRLIM
PVTHARLIM
PVTSHUT
PVTLRLIM
PVTLALIM
PVTTHRC4
Register 1
Address of TQE
PFT
PFTWHOSE
IEAPDSBL
1 Determine whether any pertinent counts (reclaim, read, swap-in) have been reset during the last paging task disablement interval. If so,
1A Set up counts for testing next interval.
Timer Second-Level Interruption Handler
PVT
PVTRECSV=PVTNPREC
PVTADRSV=PVTNPIN
PVTSINSV=PVTSPIN
TQE
TQEVAL=PVTIMEAD
2 Compare the interval reclaim count with the interval reclaim high threshold and the adjusted interval read count with its threshold.
If both are equal to or exceed their thresholds
Dispatcher
IEAODS1
Shut down a task.
3.17
If the shutdown is successful and a TCB is known
A
SCANPFTQ
Turn off reference bits for this job-step TCB's pages.
2
PVT
A
PVTSHUT=PVTSHUT+1
PVTTHRTD=1
If the shutdown is unsuccessful or the TCB is not known
1A
3 If no tasks are shutdown
1A
4 Compare the interval counts with their low thresholds.
If they both do not violate them
1A
If both are lower than or equal to the threshold values, restart a task.
Dispatcher
IEAODS1
Start up a shut down task.
3.17
B
PVT
B
PVTSHUT
PVTTHRTD=0
PVTTHRC4=0
Entered by DISPINIT or CKTHRESH (both in GETPART) to check for paging threshold violations
5 If no other tasks are shut down and an initiator region allocation request was suspended because of thrashing
DISPINIT
(in GETPART)
Inform GETPART of resource availability.
6.14
Otherwise
1A
PVT
PVTTHRC4=1
Register 15
Return code=4
PVT
PVTFLAG3
IEAPCHTH
6 Determine whether any paging supervisor thresholds have been violated.
If so
If not
Register 15
Return code=0
Caller

• Diagram 5.19 Task Disablement Algorithm and Threshold Checking (Module IEAPRPLS)

NOTES	ROUTINE NAME	LABEL
1 If the interval reclaim count, the interval read count, or the interval swap-in count is negative, that system count was reset during the last interval.	IEAPDSBL	IEAPDSBL
2 A 0 is passed to the dispatcher in register 1 to indicate shutdown. If the return code from the dispatcher is 4, no shut-down was possible. If the return code is 0, a task was shut down and either its TCB address or a zero is in register 0. If a TCB address exists, the TCB is accessed and SCANPFTQ is called. If the TCB address is 0, some subsystem (for example TSO) took some action. SCANPFTQ is called successively to scan the active page queues (PFTAC1QN, PFTAC2QN, PFTAC3QN, PFTAC4QN) and the hold page queue (PFTHQN).		ADJ
4 A 4 is passed to the dispatcher in register 1 to indicate startup. If the dispatcher is successful, PVTSHUT is decrement-ed. If it is unsuccessful, PVTSHUT is set to zero.		
5		ZEROSD
6 If PVTFLAG3≠0, a threshold has been violated.	IEAPCHTH	IEAPCHTH
$\boxed{1}$ See the table following these notes for a description of the fields in the PVT used in the Task Disable Algorithm.		
$\boxed{2}$ SCANPFTQ compares the input TCB address with the PFTWHOSE field of each PFT on the queue. For each match found, the reference bits in the page represented by that PFTE are turned off by means of an RRB instruct-ion.	SCANPFTQ	SCANPFTQ

• **Diagram 5.20**
Page Service Interface Routine

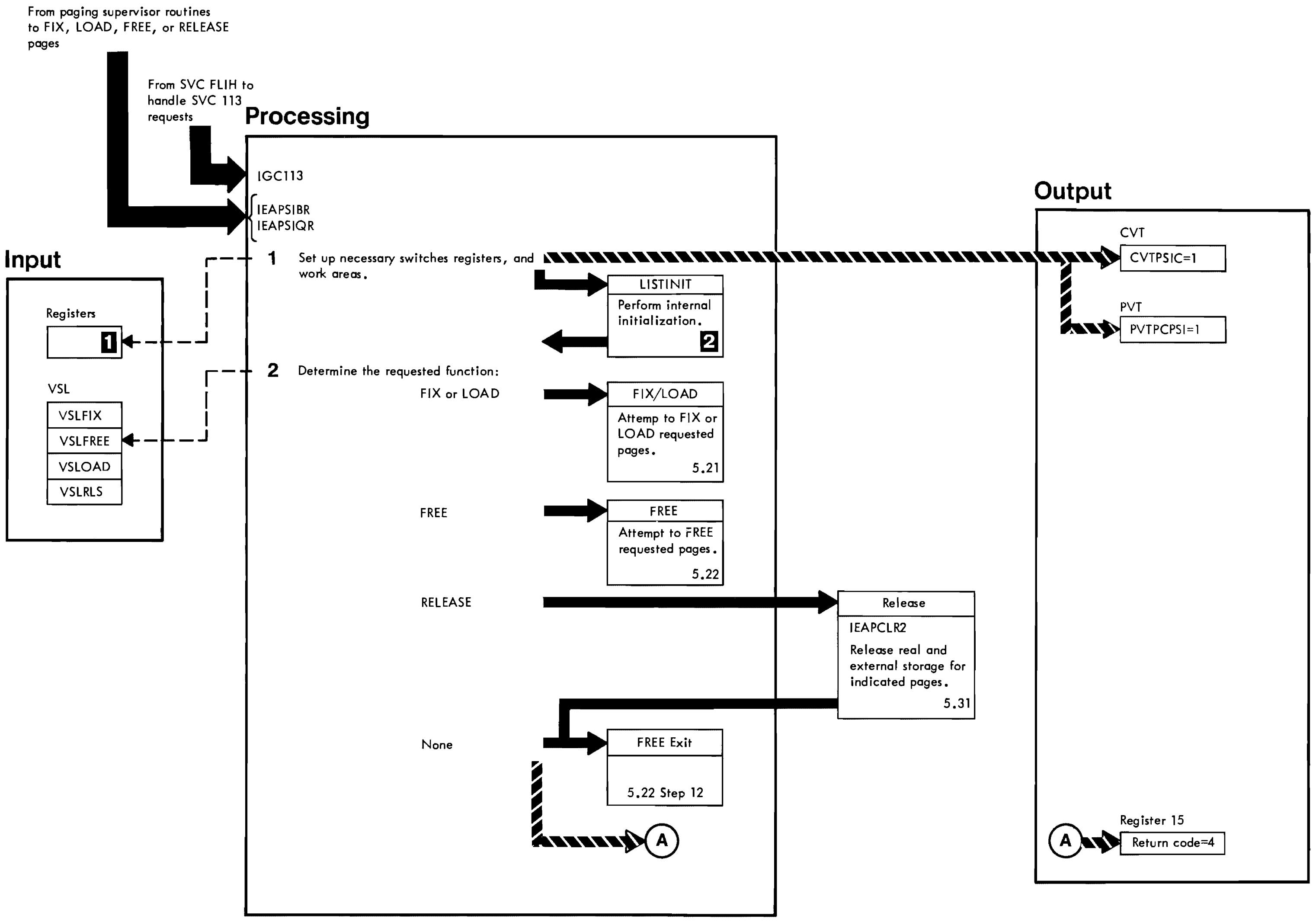

From paging supervisor routines to FIX, LOAD, FREE, or RELEASE pages
From SVC FLIH to handle SVC 113 requests
Processing
IGC113
IEAPSIBR
IEAPSIQR
Input
Registers
1
VSL
VSLFIX
VSLFREE
VSLOAD
VSLRLS
1 Set up necessary switches registers, and work areas.
LISTINIT
Perform internal initialization.
2
2 Determine the requested function:
FIX or LOAD
FIX/LOAD
Attemp to FIX or LOAD requested pages.
5.21
FREE
FREE
Attempt to FREE requested pages.
5.22
RELEASE
Release
IEAPCLR2
Release real and external storage for indicated pages.
5.31
None
FREE Exit
5.22 Step 12
A
Output
CVT
CVTPSIC=1
PVT
PVTPCPSI=1
A
Register 15
Return code=4

Diagram 5.20 Page Service Interface Routine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
1 For entry at IEAPSIBR and IEAPSIQR, the contents of registers 0 and 4 are exchanged. Entry switches are set at all entry points.	IGC113 IEAPSIBR IEAPSIQR	COMENTRY
2 If the Release routine is called, its return code is passed to the caller.		
1 Registers at entry to IGC113 are: Register 0 - ECB address (0 for FREE requests) Register 1 - List entry (VSL) or pointer to parameter list Register 2 - Second half of VSL or (for list form) irrelevant Register 4 - Requester's TCB address	IGC113	
Registers at entry to IEAPSIBR and IEAPSIQR are: Register 0 - Requester's TCB address Register 1 - VSL address (list form) or first word of VSL Register 2 - Irrelevant (list form) or second half of VSL	IEAPSIBR and IEAPSIQR	
2 LISTINIT initializes internal fields associated with the parameter list and makes register requests appear like list requests for compatability.	LISTINIT	

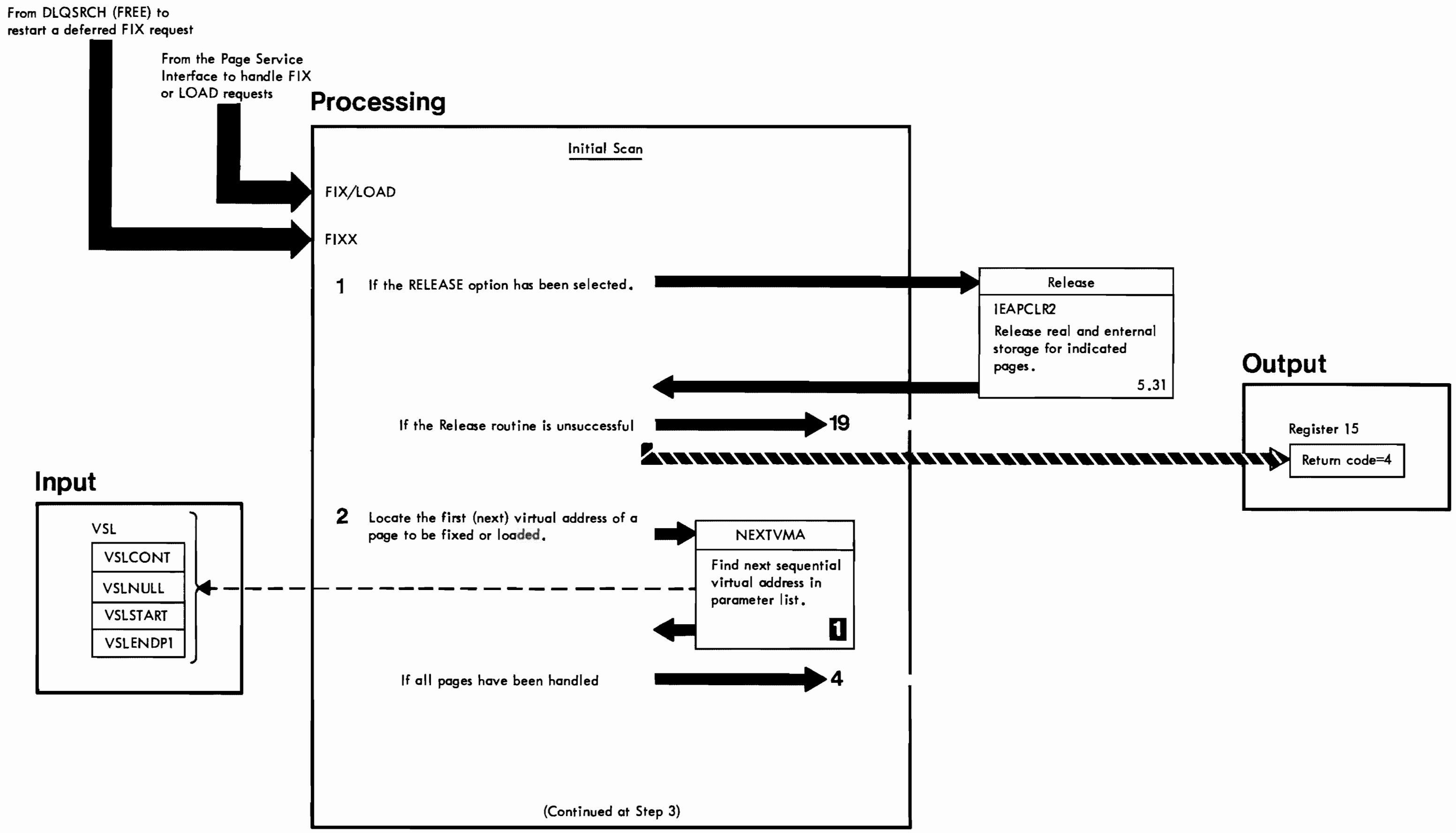
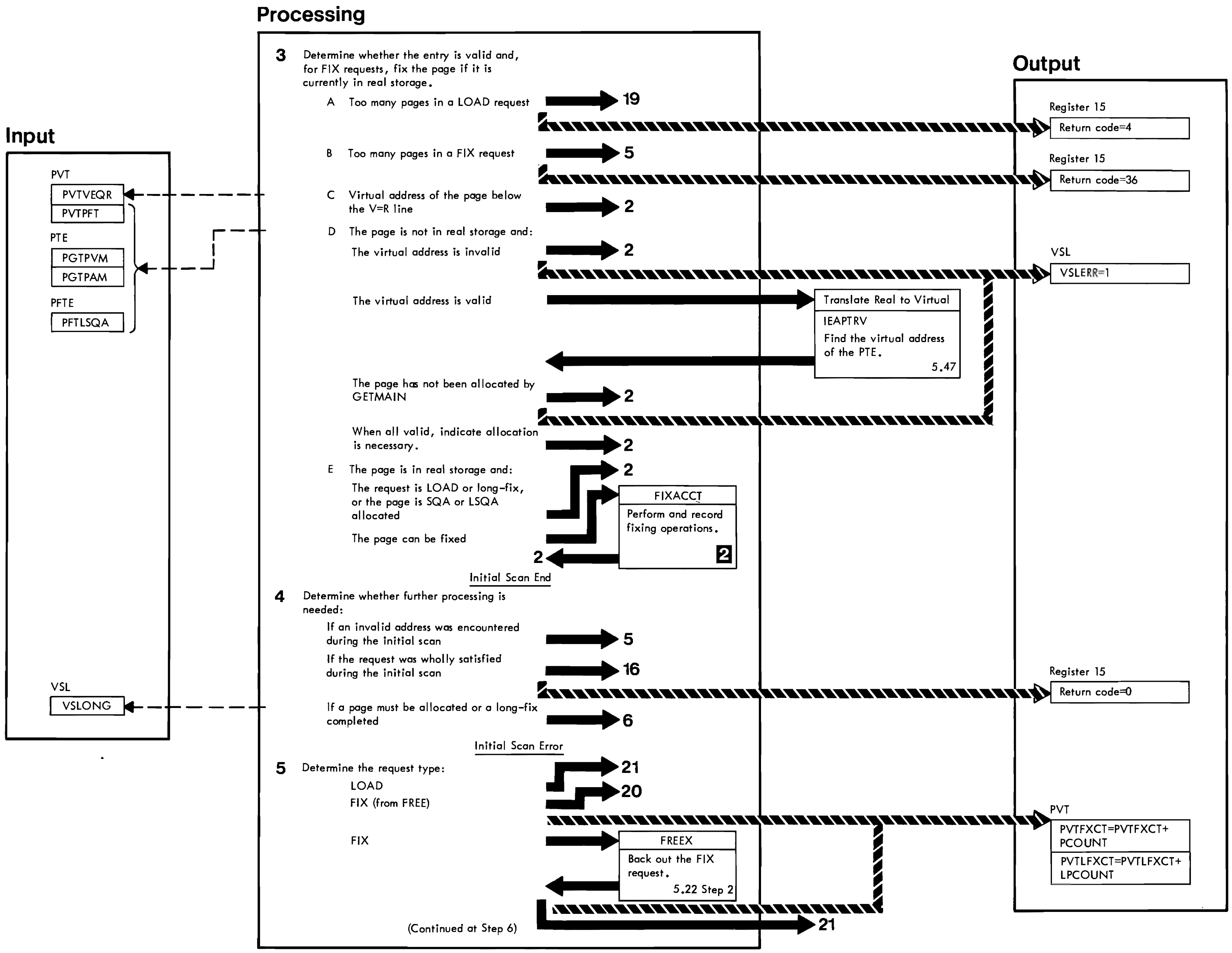

From DLQSRCH (FREE) to restart a deferred FIX request
From the Page Service Interface to handle FIX or LOAD requests
Processing
Initial Scan
FIX/LOAD
FIXX
1 If the RELEASE option has been selected.
Release
IEAPCLR2
Release real and external storage for indicated pages.
5.31
If the Release routine is unsuccessful
19
Output
Register 15
Return code=4
Input
VSL
VSLCONT
VSLNULL
VSLSTART
VSLENDP1
2 Locate the first (next) virtual address of a page to be fixed or loaded.
NEXTVMA
Find next sequential virtual address in parameter list.
1
If all pages have been handled
4
(Continued at Step 3)

Diagram 5.21 (Steps 1-2) FIX/LOAD Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
1	FIX/LOAD	FIXLOAD
For entry at FIXX, an environment for FIX/LOAD that is compatible with other entries must be established. An internal "ROOTPTR" is set to the root PCB address, register 1 is set to its contents at original entry (from PCBWK2), register 4 is set to the original TCB address (from PCBRTCB), and, for an SVC entry, register 0 is set to point to the ECB (from PCBRWK1).	FIXX	FIXX
2		NEXTVALP
1 NEXTVMA finds the next entry that is neither a null entry nor for continuation and locates the virtual address associated with it. If one is found, the return code is zero. If the end of the list has been reached, the return code is 4.	NEXTVMA	


Input
PVT
PVTVEQR
PVTPFT
PTE
PGTPVM
PGTPAM
PFTE
PFTLSQA
VSL
VSLONG
Processing
3 Determine whether the entry is valid and, for FIX requests, fix the page if it is currently in real storage.
A Too many pages in a LOAD request
19
B Too many pages in a FIX request
5
C Virtual address of the page below the V=R line
2
D The page is not in real storage and:
The virtual address is invalid
2
The virtual address is valid
Translate Real to Virtual
IEAPTRV
Find the virtual address of the PTE.
5.47
The page has not been allocated by GETMAIN
2
When all valid, indicate allocation is necessary.
2
E The page is in real storage and:
The request is LOAD or long-fix, or the page is SQA or LSQA allocated
2
The page can be fixed
FIXACCT
Perform and record fixing operations.
2
2
Initial Scan End
4 Determine whether further processing is needed:
If an invalid address was encountered during the initial scan
5
If the request was wholly satisfied during the initial scan
16
If a page must be allocated or a long-fix completed
6
Initial Scan Error
5 Determine the request type:
LOAD
21
FIX (from FREE)
20
FIX
FREEX
Back out the FIX request.
5.22 Step 2
(Continued at Step 6)
21
Output
Register 15
Return code=4
Register 15
Return code=36
VSL
VSLERR=1
Register 15
Return code=0
PVT
PVTFXCT=PVTFXCT+ PCOUNT
PVTLFXCT=PVTLFXCT+ LPCOUNT

• Diagram 5.21 (Steps 3-5) FIX/LOAD Subroutine (Module IEAPSI)

NOTES			ROUTINE NAME	LABEL
3	A & B	An internal "fix/load counter" is incremented after each return from NEXTVMA. If the counter exceeds 255, too many pages are in this request.	FIX/LOAD	
	C	If the virtual address is less than PVTVEQR, the page is below the V=R line.		NOEXCESS
	D	The virtual address is used as the argument of an LRA instruction. If the page is not in real storage and the virtual address is invalid, an error flag is set in the VSL (VSLERR=1) and a "bad address" internal flag is set to prevent the second scan from being made. If the LRA shows the virtual address to be valid but PGTPVM=1, the address returned by the LRA instruction (the real address of the PTE associated with the virtual address) is passed to IEAPTRV. If PGTPAM=0 in the PTE whose address is returned, the page has not been allocated by GETMAIN. VSLERR and the internal "bad address" switch are set. If all criteria are met, an internal "second-scan" switch is set to indicate that at least one page-in must be initiated to satisfy the request.		
	E	Long-fix processing is done during the second scan. The PFTE address is calculated as follows: Bits 8-19 of the real address from LRA x 16 +PVTPFTP If PFTLSQA=1 in the indicated PFTE, no fixing need be done.		
4				ENDLIST1
2		For a long-fix request, PFTINGFX is set to 1, and the internal "long-fix" counter (LPCOUNT) is updated if the page was not previously long-fixed. If the newly long-fixed PFTE is on a queue, it is dequeued by calling IEAPRLS3. For any request, the FOE count (for an SVC entry) via FOEON is updated, the internal "fix counter" (PCCUNT) is incremented if the page was not previously fixed, and PFTFXCT is incremented (for a branch entry or if a new FOE was built).	FIXACCT	

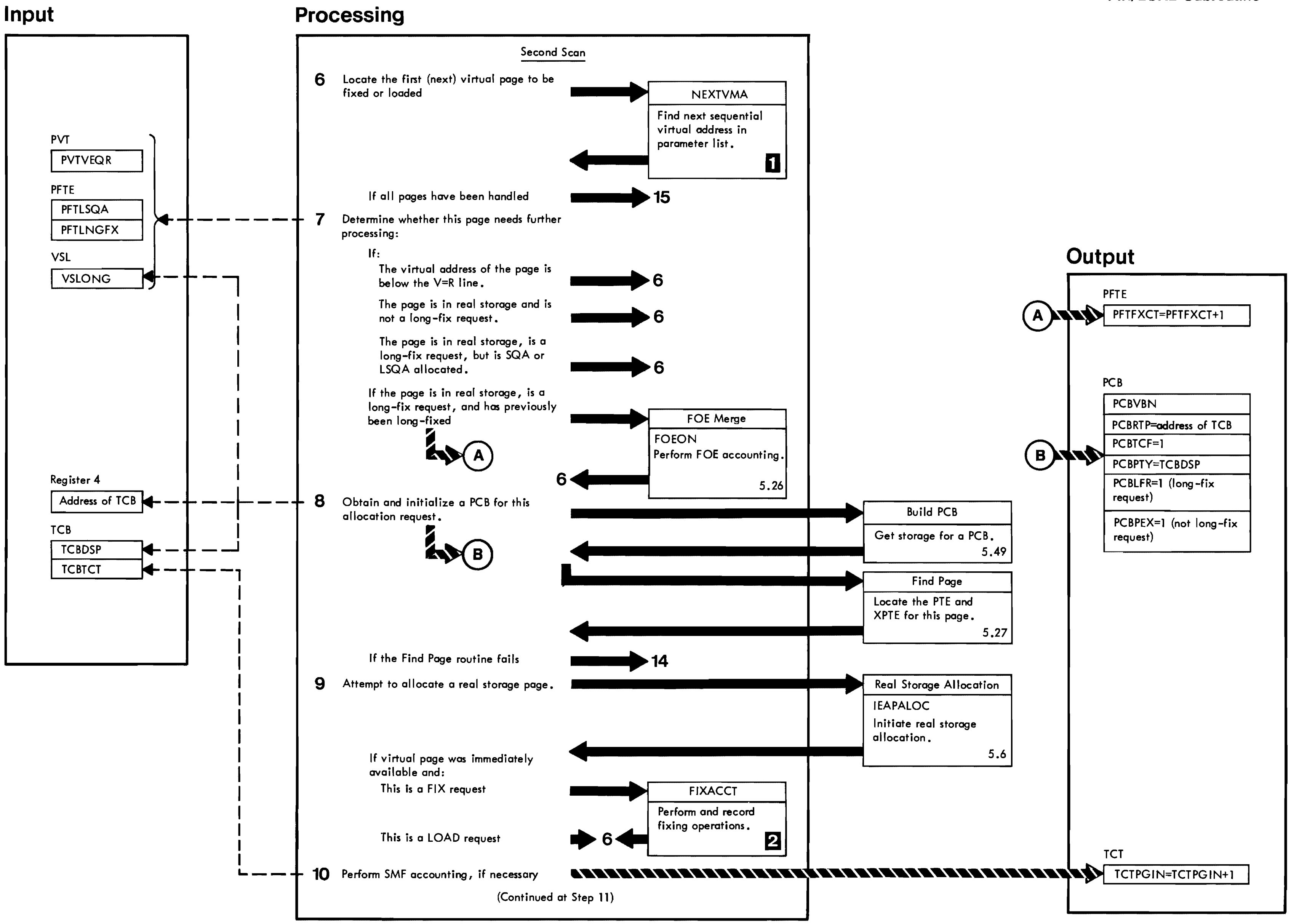
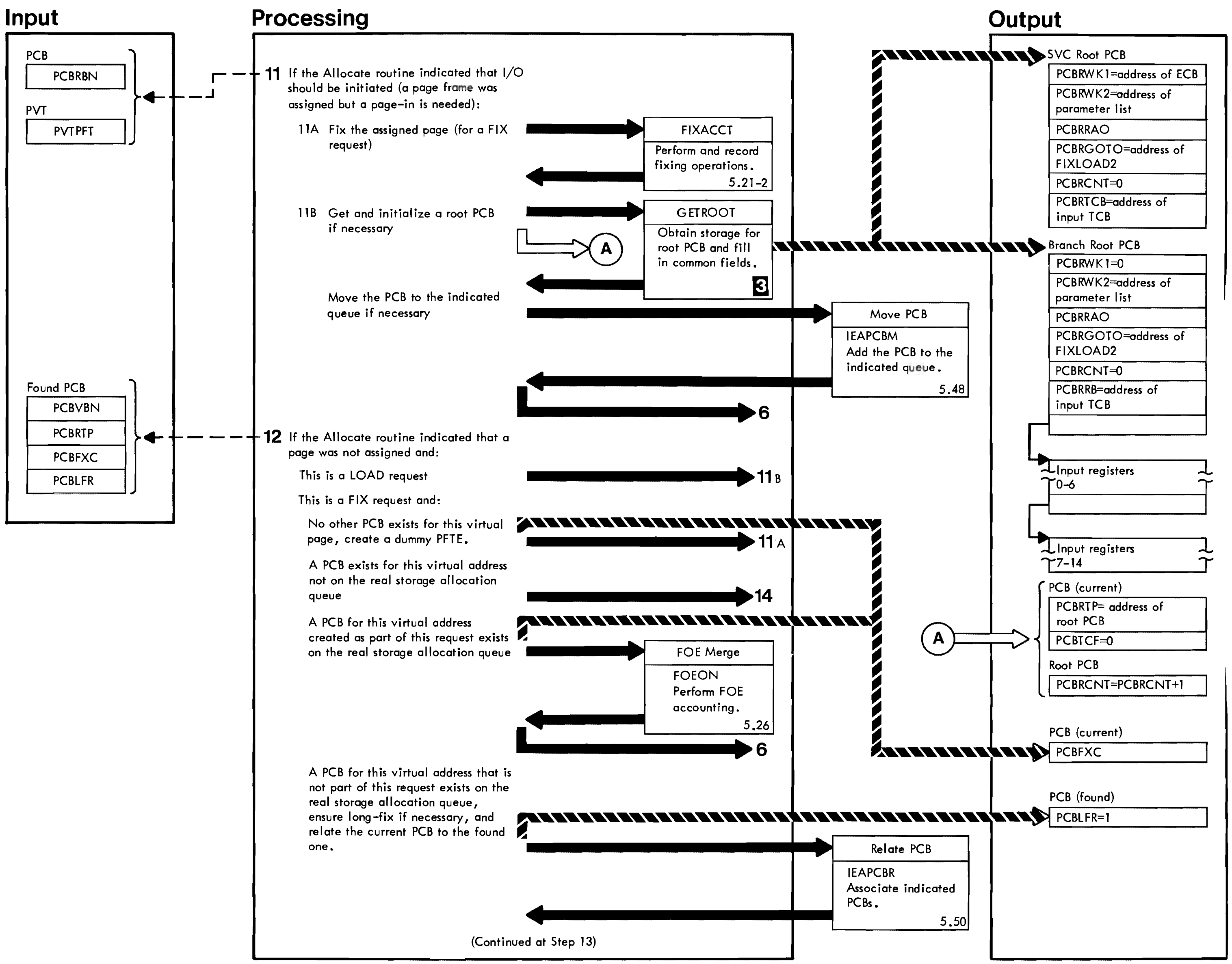

Input
Processing
Output
Second Scan
6 Locate the first (next) virtual page to be fixed or loaded
NEXTVMA
Find next sequential virtual address in parameter list.
1
If all pages have been handled
15
7 Determine whether this page needs further processing:
If:
The virtual address of the page is below the V=R line.
6
The page is in real storage and is not a long-fix request.
6
The page is in real storage, is a long-fix request, but is SQA or LSQA allocated.
6
If the page is in real storage, is a long-fix request, and has previously been long-fixed
A
FOE Merge
FOEON
Perform FOE accounting.
5.26
6
8 Obtain and initialize a PCB for this allocation request.
B
Build PCB
Get storage for a PCB.
5.49
Find Page
Locate the PTE and XPTE for this page.
5.27
If the Find Page routine fails
14
9 Attempt to allocate a real storage page.
Real Storage Allocation
IEAPALOC
Initiate real storage allocation.
5.6
If virtual page was immediately available and:
This is a FIX request
FIXACCT
Perform and record fixing operations.
2
This is a LOAD request
6
10 Perform SMF accounting, if necessary
(Continued at Step 11)
PVT
PVTVEQR
PFTE
PFTLSQA
PFTLNGFX
VSL
VSLONG
Register 4
Address of TCB
TCB
TCBDSP
TCBTCT
PFTE
A
PFTFXCT=PFTFXCT+1
PCB
PCBVBN
PCBRTP=address of TCB
B
PCBTCF=1
PCBPTY=TCBDSP
PCBLFR=1 (long-fix request)
PCBPEX=1 (not long-fix request)
TCT
TCTPGIN=TCTPGIN+1

Diagram 5.21 (Steps 6-10) FIX/LOAD Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
6 Internal pointers are reinitialized and an internal pointer to a gotten PCB (PCBPTR) is set to zero before the first call to NEXTVMA.	FIX/LOAD	NONINCOR
7 If the return code from FOEON is 4, PFTFXCT is incremented before the next call to NEXTVMA. If the return code ≠ 4, FOE accounting completed the transaction.		
8 If PCBPTR≠0, a PCB is "left over" from a previous iteration. The call to BUILDPCB is skipped and the pointed-to PCB is used.		TESTPCB
9 If the reutrn code from the Real Storage Allocation routine is zero, a page frame is immediately available and the virtual page was either reclaimed or no page-in was needed. If the return code from the Real Storage Allocation routine is 4 or 8, a page-in will be required. If the return code from the Real Storage Allocation routine is 12, the request has been related to an ongoing request.		GOTOALOC
10 If TCBTCT≠0, the TCTPGIN field of the indicated TCT is incremented to indicate that this task generated a page-in. If the return code from the Real Storage Allocation routine is 12, SMF accounting is not performed.		TESTTCT



Input

PCB
- PCBRBN

PVT
- PVTPFT

Found PCB
- PCBVBN
- PCBRTP
- PCBFXC
- PCBLFR

Processing

11 If the Allocate routine indicated that I/O should be initiated (a page frame was assigned but a page-in is needed):

11A Fix the assigned page (for a FIX request)

FIXACCT
Perform and record fixing operations.
5.21-2

11B Get and initialize a root PCB if necessary

GETROOT
Obtain storage for root PCB and fill in common fields.
3

Move the PCB to the indicated queue if necessary

Move PCB
IEAPCBM
Add the PCB to the indicated queue.
5.48

6

12 If the Allocate routine indicated that a page was not assigned and:

This is a LOAD request 11B

This is a FIX request and:

No other PCB exists for this virtual page, create a dummy PFTE. 11A

A PCB exists for this virtual address not on the real storage allocation queue 14

A PCB for this virtual address created as part of this request exists on the real storage allocation queue

FOE Merge
FOEON
Perform FOE accounting.
5.26

6

A PCB for this virtual address that is not part of this request exists on the real storage allocation queue, ensure long-fix if necessary, and relate the current PCB to the found one.

Relate PCB
IEAPCBR
Associate indicated PCBs.
5.50

(Continued at Step 13)

Output

SVC Root PCB
- PCBRWK1=address of ECB
- PCBRWK2=address of parameter list
- PCBRRAO
- PCBRGOTO=address of FIXLOAD2
- PCBRCNT=0
- PCBRTCB=address of input TCB

Branch Root PCB
- PCBRWK1=0
- PCBRWK2=address of parameter list
- PCBRRAO
- PCBRGOTO=address of FIXLOAD2
- PCBRCNT=0
- PCBRRB=address of input TCB

Input registers 0-6

Input registers 7-14

PCB (current)
- PCBRTP= address of root PCB
- PCBTCF=0

Root PCB
- PCBRCNT=PCBRCNT+1

PCB (current)
- PCBFXC

PCB (found)
- PCBLFR=1

Diagram 5.21 (Steps 11-12) FIX/LOAD Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
11 This processing is done if the return code from the Real Storage Allocation routine was 4 or 12.	FIX/LOAD	CALLFIXA
The PFTE address is calculated by adding PCBRBN to PVTPFTP and is passed to FIXACCT.		
An internal ROOTPTR is tested and, if zero, GETROOT is called. If ROOTPTR≠0, the root PCB it points to is used.	GETROOT	
3 GETROOT obtains storage for a root PCB (via Build PCB) and initializes it. The type of root PCB obtained depends on the entry type.		
The return code from Real Storage Allocation is tested for 12 (PCB related). If it is 12, either Real Storage Allocation set it or it was set by this routine after the call to the RELATE PCB routine, and the call to the Move PCB routine is bypassed.		
The internal PCBPTR is zeroed to indicate that the PCB just used cannot be re-used before the next call to NEXTVMA.		
12 This processing is done if the return code from the Real Storage Allocation routine was 8.		
If XPTPCBQ=0, no other PCB exists for the virtual address specified in this request. The fix count in the current PCB (PCBFXC) is set to one and FIXACCT is called with a pointer to a dummy PFTE whose PFTFXCT field=0, and whose PFTVBN is set to the appropriate virtual address.		NEXTTST1
If XPTPCBQ≠0, the Real Storage Allocation queue is scanned for a PCB whose PCBVBN field matches PCBVBN in the current PCB. If a match is not found, the error exit is taken.		
If PCBRTP=ROOTPTR in the found PCB, this PCB is part of this request and FOE accounting is performed. If the return code from FOEON is 4, PCBFXC is incremented.		
If there was no root PCB for this request (ROOTPTR=0), or the root PCB pointers did not match, the current PCB is related to the found PCB via the Relate PCB routine. If the request is for a long-fix and the found PCB is for a fix but not a long-fix (PCBFXC≠0 and PCBLFR=0), PCBLFR is turned on and LPCOUNT is incremented before the call to the Relate PCB routine.		

Diagram 5.21 (Steps 13-17)
FIX/LOAD Subroutine

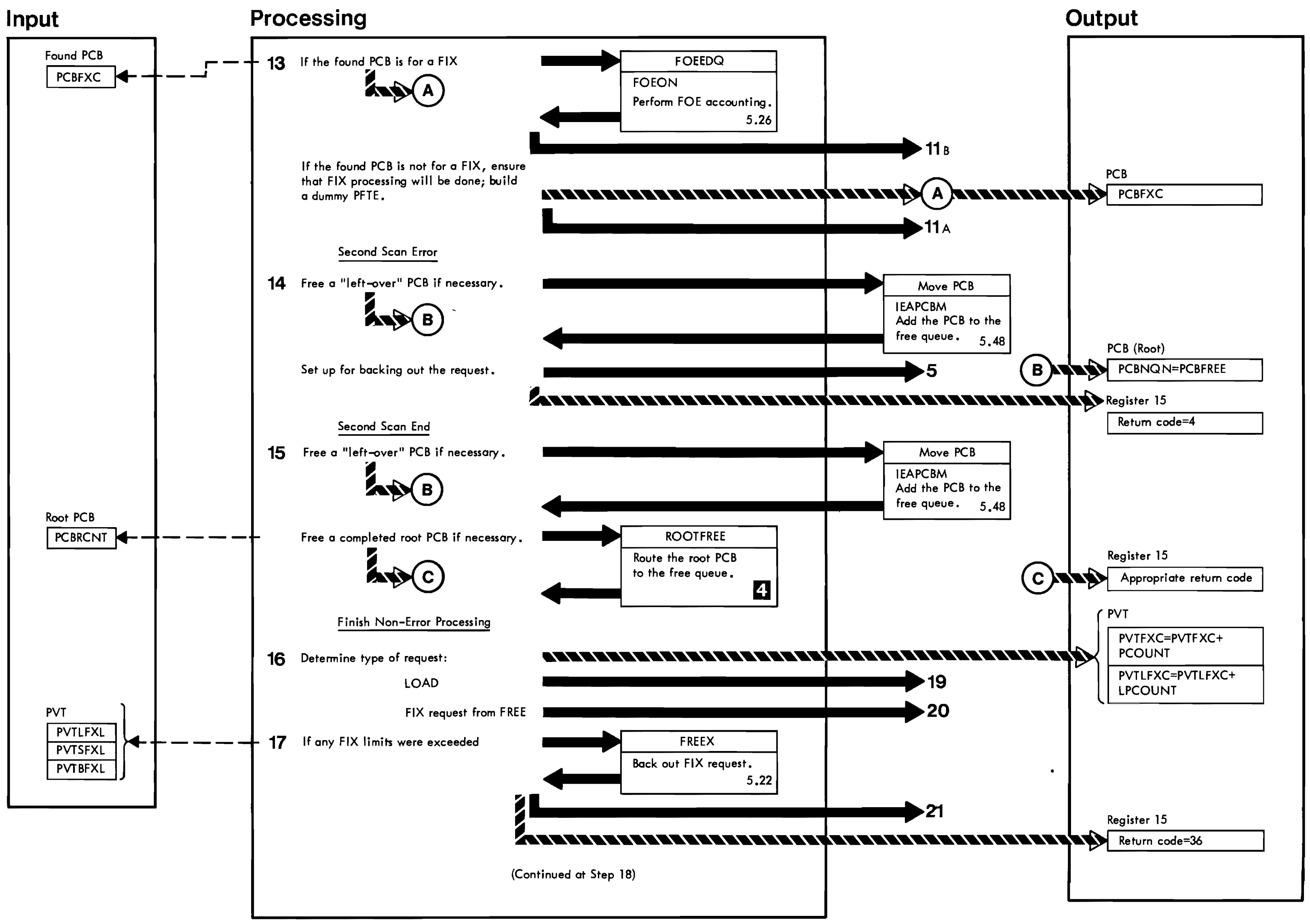

Input
Processing
Output
Found PCB
PCBFXC
Root PCB
PCBRCNT
PVT
PVTLFXL
PVTSFXL
PVTBFXL
13 If the found PCB is for a FIX
A
FOEEDQ
FOEON
Perform FOE accounting.
5.26
11 B
If the found PCB is not for a FIX, ensure that FIX processing will be done; build a dummy PFTE.
A
11 A
Second Scan Error
14 Free a "left-over" PCB if necessary.
B
Move PCB
IEAPCBM
Add the PCB to the free queue.
5.48
Set up for backing out the request.
5
Second Scan End
15 Free a "left-over" PCB if necessary.
B
Move PCB
IEAPCBM
Add the PCB to the free queue.
5.48
Free a completed root PCB if necessary.
C
ROOTFREE
Route the root PCB to the free queue.
4
Finish Non-Error Processing
16 Determine type of request:
LOAD
19
FIX request from FREE
20
17 If any FIX limits were exceeded
FREEX
Back out FIX request.
5.22
21
(Continued at Step 18)
PCB
PCBFXC
B
PCB (Root)
PCBNQN=PCBFREE
Register 15
Return code=4
C
Register 15
Appropriate return code
PVT
PVTFXC=PVTFXC+PCOUNT
PVTLFXC=PVTLFXC+LPCOUNT
Register 15
Return code=36

Diagram 5.21 (Steps 13-17) FIX/LOAD Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
13 If the return code from FOEON is 4, PCBFXC is incremented by 1.	FIX/LOAD	
If the fix count (PCBFXC) in the found PCB (the one to which the current request was related) is zero (PCB not for fix), it is set to 1 and a dummy PFTE (see Note 12) is set up before the call to FIXACCT.		
14 If the internal PCBPTR≠0, the PCB it points to is routed to the free queue (PCBNQN=PCBFREE).		ERROREX1
An internal "synthetic end" switch and pointers to the current and last VSLs processed are set up so that only the portion of the request that was processed thus far will be backed out by FREEX.		
15 A possible leftover PCB is handled as in Note 14.		LISTEND
If the internal ROOTPTR=0, none of the requested pages require further paging action. If ROOTPTR≠0, the indicated root PCB does not represent any pending requests (PCBRCNT=0), and the caller of FIX/LOAD was not FREE, the root PCB is disposed of.		
16 The return code is set to 8 if a root PCB remains, or to 0 if the operation is complete.		
PCOUNT (newly fixed pages) and LPCOUNT (newly long-fixed pages) are added to PVTFXC and PVTLFXC respectively to keep count of fixed pages in the system before the request type is determined.		FINISH
17 For a long-fix request, if LPCOUNT>PVTLFXL, the long-fix limit has been exceeded.		
If PCOUNT>PVTSFXL (SVC request) or PVTBFXL (Branch request), the fix limit has been exceeded.		
4 ROOTFREE returns the root PCB to the free queue (via MOVEPCB) as follows: • For an SVC root - Zero the root and set PCBNQN= PCBFREE. • For a Branch root - Zero the root, chain the three blocks together by PCBFQP and PCBBQR, and set PCBNQN of the first block equal to PCBFREE.	ROOTFREE	

Diagram 5.21 (Step 18)
FIX/LOAD Subroutine

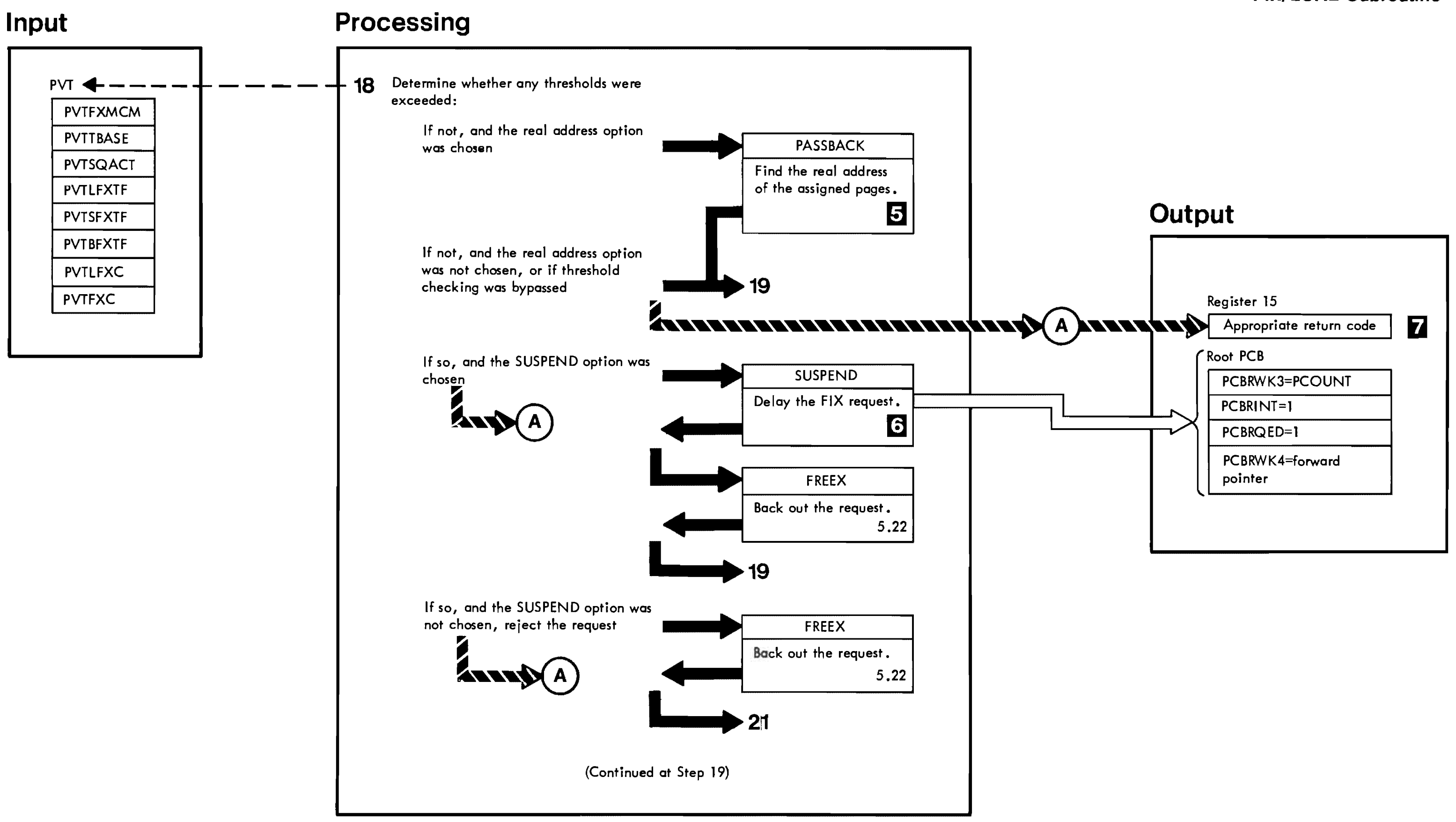

Input
PVT
PVTFXMCM
PVTTBASE
PVTSQACT
PVTLFXTF
PVTSFXTF
PVTBFXTF
PVTLFXC
PVTFXC
Processing
18 Determine whether any thresholds were exceeded:
If not, and the real address option was chosen
PASSBACK
Find the real address of the assigned pages.
5
If not, and the real address option was not chosen, or if threshold checking was bypassed
19
A
If so, and the SUSPEND option was chosen
A
SUSPEND
Delay the FIX request.
6
FREEX
Back out the request.
5.22
19
If so, and the SUSPEND option was not chosen, reject the request
A
FREEX
Back out the request.
5.22
21
(Continued at Step 19)
Output
Register 15
Appropriate return code
7
Root PCB
PCBRWK3=PCOUNT
PCBRINT=1
PCBRQED=1
PCBRWK4=forward pointer

Diagram 5.21 (Step 18) FIX/LOAD Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
18 For a long-fix request: If PVTFXMCM=1, threshold checking should be bypassed because this is a DSS activate request. If PVTLFXC>PVTTBASE-(PVTSQACT+PVTLFXTF), the threshold has been exceeded. For an SVC request, if PVTFXC>PVTTBASE-(PVTSQACT+PVTSFXTF), the threshold has been exceeded. For a Branch request, PVTBFXTF is used instead of PVTSFXTF in the above equation. PASSBACK is called only if the request has thus far completed normally (return code =0).	FIX/LOAD	CHECKLIM
5 PASSBACK satisfies the real address option by using the virtual address of the indicated page (bytes 1-3 of a normat VSL) as the argument of an LRA instruction and placing the resulting real address in bytes 5-7 of the VSL.	PASSBACK	
6 SUSPEND queues a FIX request that cannot be honored because of a threshold violation as follows: • Obtain a root PCB (via GETROOT) if one does not already exist and mark it (or the previously existing root) for interception and as queued (PCBRINT=1 and PCBRQED=1 respectively). • For an SVC entry, add it to the SVC FIX delay queue (PVTSFXDQ) in the proper place. • For a branch entry, add the Root to the Branch Entry FIX delay queue (PVTBFXDQ) in the proper place. • Root PCBs are queued by number of pages needed, from lowest to highest.	SUSPEND	

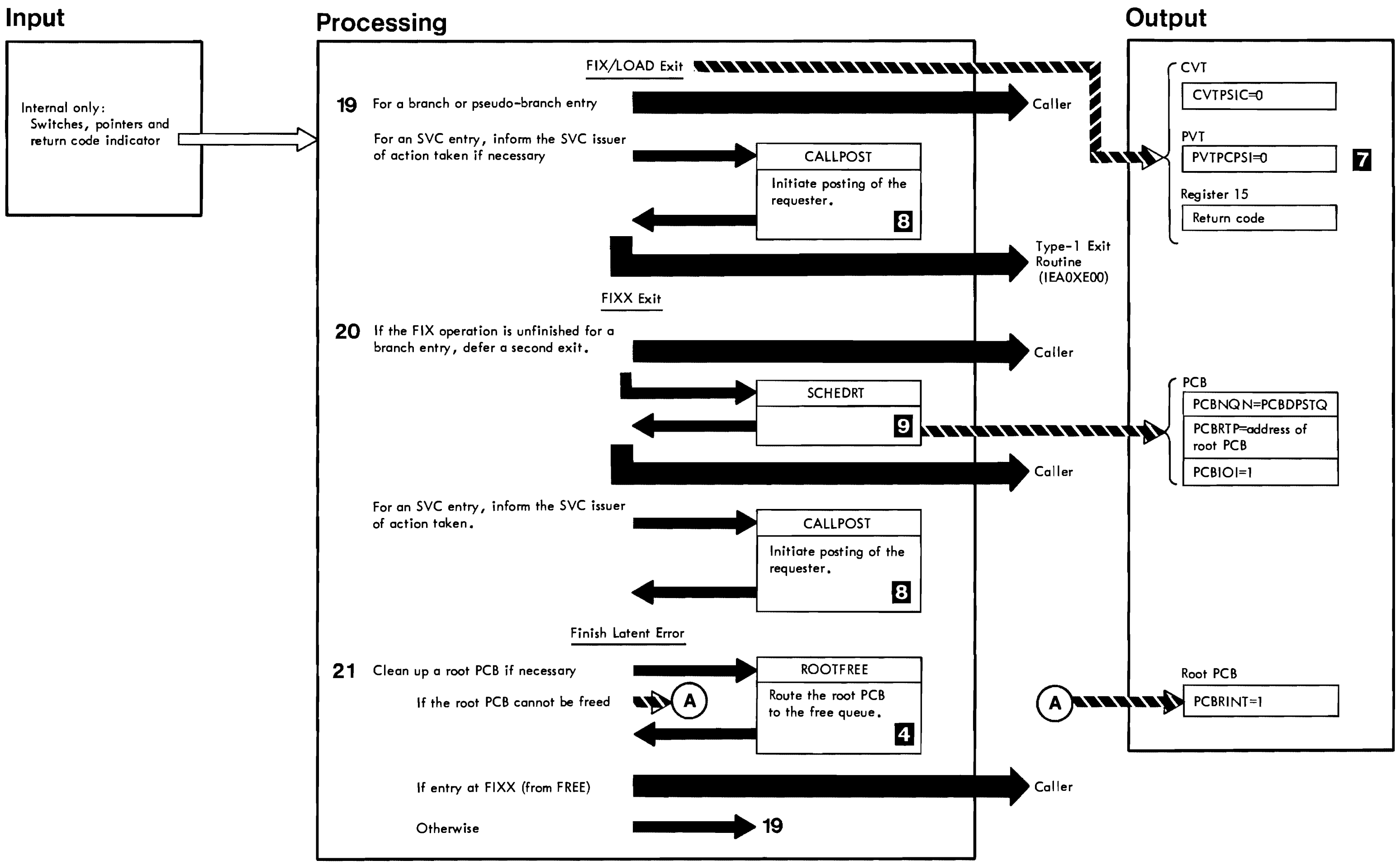

Input
Internal only:
Switches, pointers and return code indicator
Processing
FIX/LOAD Exit
19 For a branch or pseudo-branch entry
Caller
For an SVC entry, inform the SVC issuer of action taken if necessary
CALLPOST
Initiate posting of the requester.
8
Type-1 Exit Routine (IEA0XE00)
FIXX Exit
20 If the FIX operation is unfinished for a branch entry, defer a second exit.
Caller
SCHEDRT
9
Caller
For an SVC entry, inform the SVC issuer of action taken.
CALLPOST
Initiate posting of the requester.
8
Finish Latent Error
21 Clean up a root PCB if necessary
ROOTFREE
Route the root PCB to the free queue.
4
If the root PCB cannot be freed
A
If entry at FIXX (from FREE)
Caller
Otherwise
19
Output
CVT
CVTPSIC=0
PVT
PVTPCPSI=0
7
Register 15
Return code
PCB
PCBNQN=PCBDPSTQ
PCBRTP=address of root PCB
PCBIOI=1
A
Root PCB
PCBRINT=1

Diagram 5.21 (Steps 19-21) FIX/LOAD Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
19 CALLPOST is called only when return codes 0, 4, 36, or 44 (when the SUSPEND option is not specified) are indicated.	FIX/LOAD	FLEXIT
20 If the indicated return code is 8, exit is made immediately to the caller (FREE). For a Branch entry, if the indicated return code is 4, it is changed to 28. The return code is placed in PCBRWK4 before SCHEDRT is called. For an SVC entry, if the indicated return code is 28, it is changed to 4 before CALLPOST is called.		FIXXEXIT
21 If there is a Root PCB and it has no more associated requests, (PCBRCNT=0), it is freed. Otherwise, it is marked for interception (PCBRINT=1).		TESTROOT
7 For explanation of the return codes and completion codes for POST, see table following notes for this diagram.		
8 CALLPOST sets up parameters for and passes control to POST at entry point IEA0PT01.	CALLPOST	
9 SCHEDRT is called to schedule a root PCB on the delayed Posting queue when a second exit is needed but must be deferred. It obtains a PCB (via Build PCB), initializes it, and adds it to the delayed posting queue (via move PCB).	SCHEDRT	

FIX/LOAD RETURN and COMPLETION CODES

FIX SVC Entry

Return Code	Meaning	Action to be Taken
0	Operation complete	None; wait on ECB if desired.
4	Error detected, no pages fixed	Error handling; wait on ECB if desired.
8	Operation proceeding	Issue WAIT macro instruction on same ECB as used in the PGFIX macro.
36	Request too large	Error handling; wait on ECB if desired.

With SUSPEND option specified:

Return Code	Meaning	Action to be Taken
40	FIX request queued due to shortage of long-term fixable real storage (long wait)	If no interlock exposure exists and if long wait acceptable, issue WAIT macro instruction. Otherwise, cancel FIX with PGFREE macro instruction specifying ECB option.
44	FIX request queued due to shortage of fixable real storage (short-term wait)	If no interlock exposure exists, issue WAIT macro instruction. Otherwise, cancel the FIX as above.

Without SUSPEND option specified:

Return Code	Meaning	Action to be Taken
44	FIX request rejected due to shortage of fixable real storage	Reissue request at a later time. Wait on ECB if desired. If no interlock exposure exists, reissue PGFIX macro instruction with SUSPEND specified.

Comp. Code	Meaning	Action to be Taken
0	Operation complete	None
4	Error detected, no pages fixed	Error handling
36	FIX request too large, no pages fixed	Error handling
44	FIX requested rejected due to shortage of fixable storage	Reissue request at a later time. If no interlock exposure exists, reissue PGFIX macro with SUSPEND specified.
48	Operation purged	Reinitialize the ECB and reissue the PGFIX macro instruction. This code occurs when an asynchronous system event has caused the issuing task to be quiesced.

LOAD SVC Entry

Return Code	Meaning	Action to be Taken
0	Operation complete	None
4	Error detected	Error handling
8	Operation proceeding	Issue WAIT macro instruction for the ECB that was used for the PGLOAD macro.

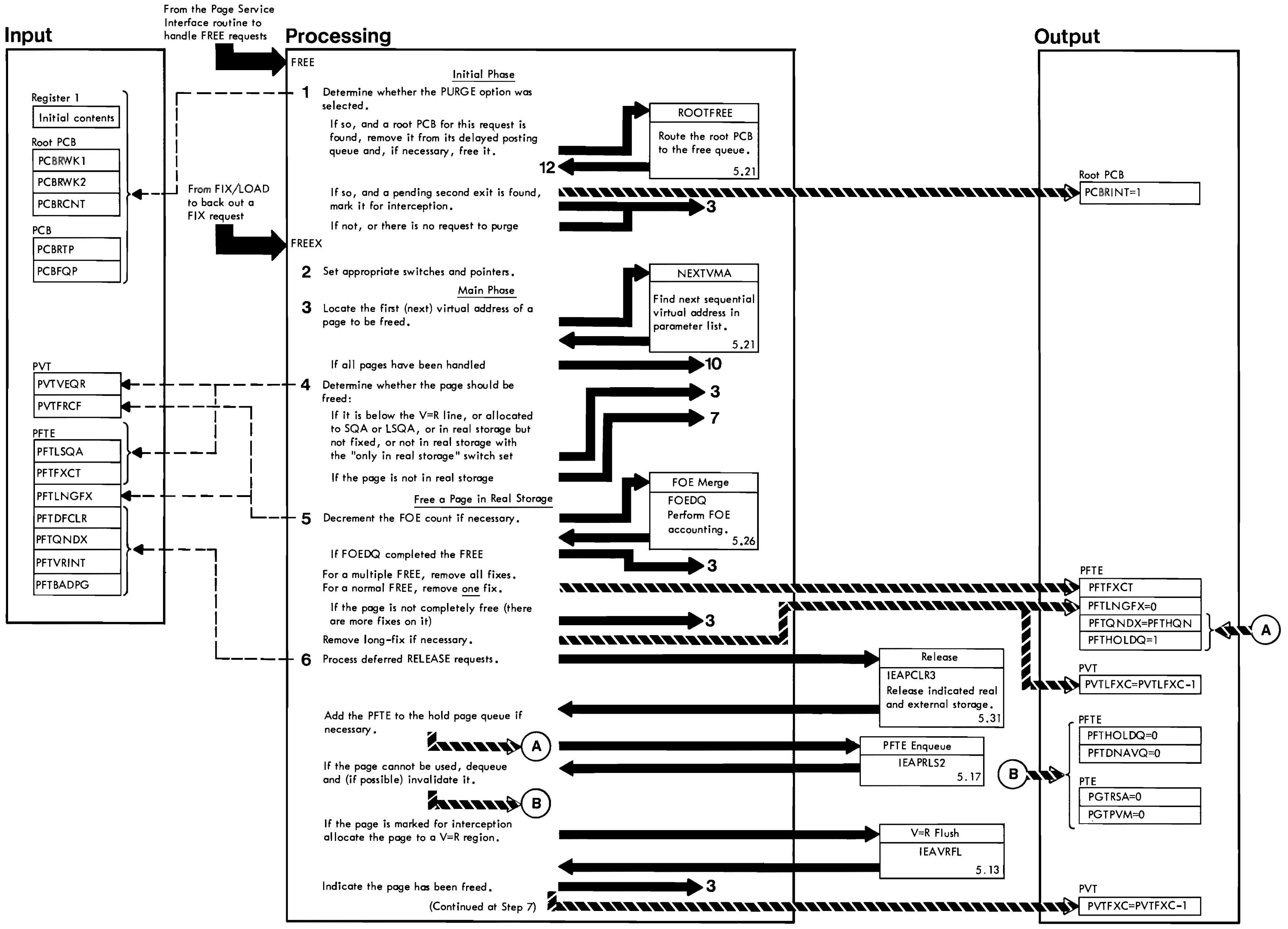
Comp. Code	Meaning	Action to be Taken
0	Operation complete	None
4	Error Detected	Error handling

FIX Branch Entry

Return Code	Meaning	Action to be Taken
0	Operation complete	None
4	Request not performed invalid address	None
8	Operation initiated but not yet complete	Exit, but leave routine reenterable.
28	Request not performed invalid address	Error handling. This return code is given on second entry to the requester ("second exit") from the FIXLOAD 2.
32	Operation complete	None. This return code is given on second entry to the requester ("second exit") from FIXLOAD 2.
36	Request too large	Error handling
40	Request queued due to shortage of long-term fixable real storage (long wait)	If no interlock exposure exists and if long wait acceptable, exit but leave routine reenterable. Otherwise, cancel FIX.
44	Request queued due to shortage of fixable real storage (short-term wait)	If no interlock exposure exists, exit but leave routine reenterable. Otherwise, cancel FIX.

LOAD Branch Entry

Return Code	Meaning	Action to be Taken
0	Operation complete	None
4	Request not performed, invalid address	Error handling
8	Operation initiated but not yet complete	Exit, but leave routine reenterable
28	Request not performed, invalid address	Error handling. This return code is given on second entry to the requester ("second exit") from FIXLOAD 2.
32	Operation complete	None. This return code is given on second entry to the requester ("second exit") from FIXLOAD 2.



Input

Register 1
- Initial contents

Root PCB
- PCBRWK1
- PCBRWK2
- PCBRCNT

PCB
- PCBRTP
- PCBFQP

PVT
- PVTVEQR
- PVTFRCF

PFTE
- PFTLSQA
- PFTFXCT
- PFTLNGFX
- PFTDFCLR
- PFTQNDX
- PFTVRINT
- PFTBADPG

From the Page Service Interface routine to handle FREE requests

From FIX/LOAD to back out a FIX request

Processing

FREE

Initial Phase

1 Determine whether the PURGE option was selected.

If so, and a root PCB for this request is found, remove it from its delayed posting queue and, if necessary, free it.

ROOTFREE — Route the root PCB to the free queue. 5.21

12

If so, and a pending second exit is found, mark it for interception. → 3

If not, or there is no request to purge

FREEX

2 Set appropriate switches and pointers.

Main Phase

3 Locate the first (next) virtual address of a page to be freed.

NEXTVMA — Find next sequential virtual address in parameter list. 5.21

If all pages have been handled → 10

4 Determine whether the page should be freed:

If it is below the V=R line, or allocated to SQA or LSQA, or in real storage but not fixed, or not in real storage with the "only in real storage" switch set → 3

If the page is not in real storage → 7

Free a Page in Real Storage

5 Decrement the FOE count if necessary.

FOE Merge — FOEDQ Perform FOE accounting. 5.26

If FOEDQ completed the FREE → 3

For a multiple FREE, remove all fixes. For a normal FREE, remove one fix.

If the page is not completely free (there are more fixes on it) → 3

Remove long-fix if necessary.

6 Process deferred RELEASE requests.

Release — IEAPCLR3 Release indicated real and external storage. 5.31

Add the PFTE to the hold page queue if necessary. → A

PFTE Enqueue — IEAPRLS2 5.17

If the page cannot be used, dequeue and (if possible) invalidate it. → B

If the page is marked for interception allocate the page to a V=R region.

V=R Flush — IEAVRFL 5.13

Indicate the page has been freed. → 3

(Continued at Step 7)

Output

Root PCB
- PCBRINT=1

PFTE
- PFTFXCT
- PFTLNGFX=0
- PFTQNDX=PFTHQN
- PFTHOLDQ=1

A

PVT
- PVTLFXC=PVTLFXC-1

B

PFTE
- PFTHOLDQ=0
- PFTDNAVQ=0

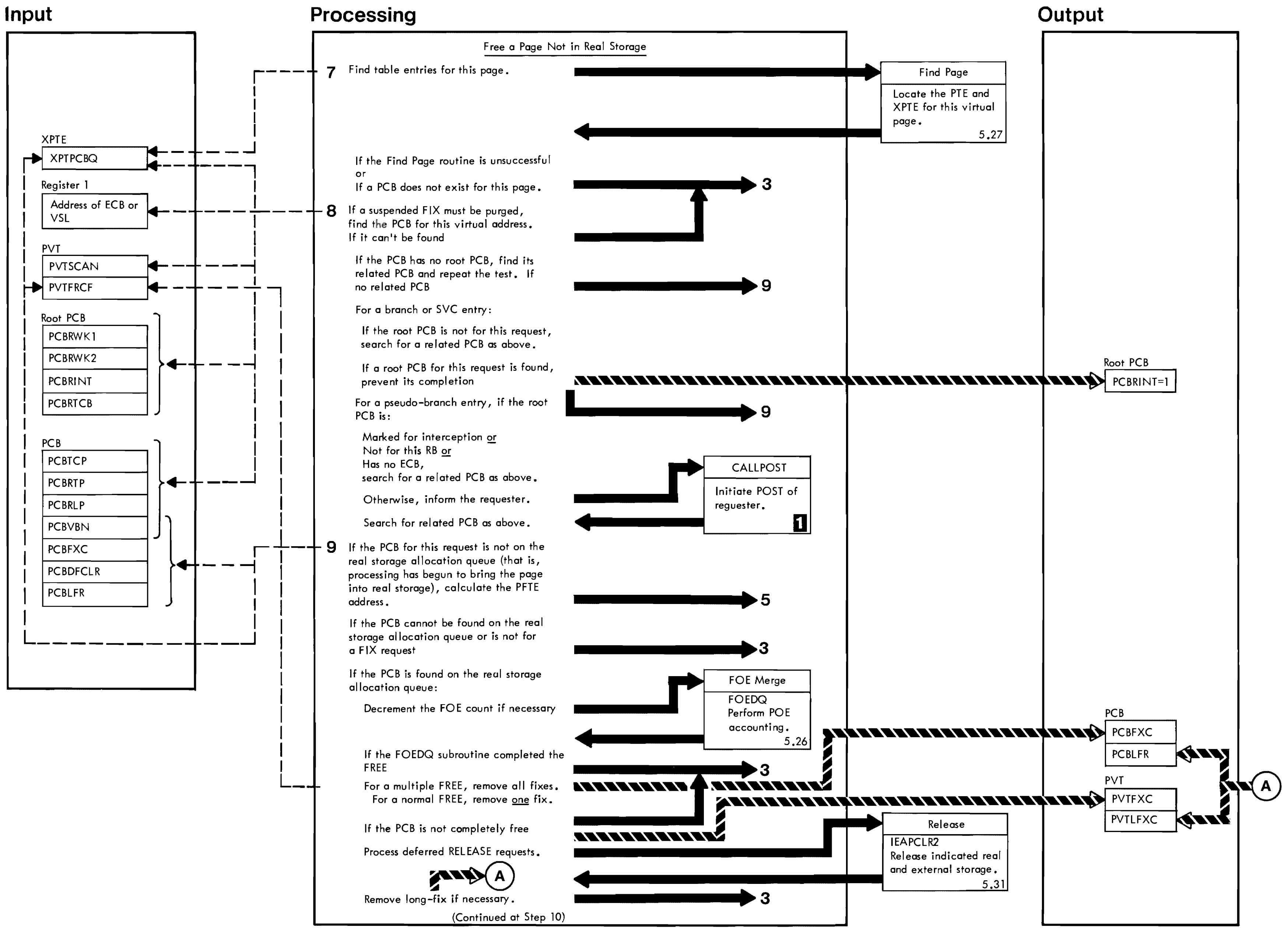
PTE
- PGTRSA=0
- PGTPVM=0

PVT
- PVTFXC=PVTFXC-1

• Diagram 5.22 (Steps 1-6) FREE Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
1 If this is an SVC entry and register 0 ≠ 0 (PURGE option specified), the SVC Delay Queue is searched for a root PCB for this request (one whose PCBRWK1 matches the ECB address in Register 1). If found, the root PCB is removed from the queue and its PCBRQED field is zeroed. If there are more requests represented by this root (PCBRCNT≠0), the call to ROOTFREE is bypassed and control goes directly to Step 12. If this is a Branch entry, list-type, and the suspend option bit is on in the VSL, processing proceeds as for an SVC entry (above), except that the Branch Delay Queue is searched, comparing PCBRWK2 and register 1. If no match is found for a Branch entry, the Delayed Post Queue (PCBDPSTQ) is searched comparing the PCBRWK2 field of each root PCB pointed to by the PCBRTP of each PCB on the queue. If a match is found, the root PCB is marked for interception.	FREE	FREE
2 FREEX is the interface between FIX/LOAD and FREE when a request must be backed out due to an error detected in FIX/LOAD. It is, essentially, FREE controlled by the following set of internal switches set by FIX: In-real-storage-only -- Only the initial scan of FIX/LOAD has completed. Only FREE pages in real storage. Synthetic-end -- Error found in the middle of the second scan of FIX. FREE pages both in and out of real storage up to the point indicated by the internal "STOP1" and "STOP2" pointers, then FREE pages in real storage only. From-FIX -- Set at entry to indicate to FREE where and how to exit.	FREEX	FREEX
3 Upon return from NEXTVMA, if end-of-list has not been indicated (return code≠4), and the "synthetic-end" switch is on, the current VSL is compared to the internal "STOP" pointers to determine whether the address is beyond the indicated point. If so, the "synthetic-end" switch is turned off and the "in-real-storage-only" switch is turned on.	FREE and FREEX	CALLNVMA

NOTES	ROUTINE NAME	LABEL
4 If the virtual address of the page to be freed is below the V=R line (PVTVEQR), the scan continues. If not, the virtual address is used as the argument of an LRA instruction. If the condition code indicates that the page is in real storage, its PFTE is calculated. If PFTLSQA=1, the page is SQA or LSAQ allocated. If PFTFXCT=0, the page is not fixed. In either case, the scan continues.		LOADREAL
5 If the return code from FOEDQ=0, the scan continues. (Note: Since PFTFXCT is not decremented, the page is still fixed.) If PVTFRCF=1, this is a multiple FREE request and PFTFXCT is zeroed. If PVTFRCF=0, PFTFXCT is decremented by one. If PFTFXCT does not now equal 0, the scan continues. If the page is now truly FREE (PFTFXCT=0), and the long-fix flag is on (PFTLNGFX=1), it is turned off and PVTLFXC is decremented.		TSQA
6 If PFTDFCLR=1, a deferred RELEASE must be processed. If the PFTE is not on a queue (PFTQNDX=0) and the page is in real storage, the PFTE is routed to the hold page queue. After the PFTE is enqueued (or if it was already on a queue, or if the page was not in real storage), PFTBADPG is tested. If the page is marked unusable by RMS (PFTBADPG=1) (see Diagram 5.18): • The PFTE is dequeued (via IEAPL3).		TCLEAR
• Find Page is called to locate the PTE and, if Find Page is successful, the page is invalidated (PGTPVM is set to 1), and a PTLB is issued to clear the DLATS (see Diagram 5.27).	IEAPRLS3 IEAPFP2	
If PFTBADPG=0, or after dequeuing the PFTE and invalidating the page, if the page is marked for V=R interception (PFTVRINT=1), V=R Flush is called to complete (or cancel if PFTBADPG=1) a delayed V=R region allocation request (see Diagram 5.13).	IEAPVRFL	



Input

XPTE
- XPTPCBQ

Register 1
- Address of ECB or VSL

PVT
- PVTSCAN
- PVTFRCF

Root PCB
- PCBRWK1
- PCBRWK2
- PCBRINT
- PCBRTCB

PCB
- PCBTCP
- PCBRTP
- PCBRLP
- PCBVBN
- PCBFXC
- PCBDFCLR
- PCBLFR

Processing

Free a Page Not in Real Storage

7 Find table entries for this page.

Find Page
Locate the PTE and XPTE for this virtual page. 5.27

If the Find Page routine is unsuccessful or
If a PCB does not exist for this page. → 3

8 If a suspended FIX must be purged, find the PCB for this virtual address.
If it can't be found → 3

If the PCB has no root PCB, find its related PCB and repeat the test. If no related PCB → 9

For a branch or SVC entry:

If the root PCB is not for this request, search for a related PCB as above.

If a root PCB for this request is found, prevent its completion

For a pseudo-branch entry, if the root PCB is: → 9

Marked for interception or
Not for this RB or
Has no ECB,
search for a related PCB as above.

Otherwise, inform the requester.

CALLPOST
Initiate POST of requester. 1

Search for related PCB as above.

9 If the PCB for this request is not on the real storage allocation queue (that is, processing has begun to bring the page into real storage), calculate the PFTE address. → 5

If the PCB cannot be found on the real storage allocation queue or is not for a FIX request → 3

If the PCB is found on the real storage allocation queue:

Decrement the FOE count if necessary

FOE Merge
FOEDQ
Perform POE accounting. 5.26

If the FOEDQ subroutine completed the FREE → 3

For a multiple FREE, remove all fixes.
For a normal FREE, remove one fix.

If the PCB is not completely free

Process deferred RELEASE requests.

Release
IEAPCLR2
Release indicated real and external storage. 5.31

A

Remove long-fix if necessary. → 3

(Continued at Step 10)

Output

Root PCB
- PCBRINT=1

PCB
- PCBFXC
- PCBLFR

PVT
- PVTFXC
- PVTLFXC

A

Diagram 5.22 (Steps 7-9) FREE Subroutine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
7 IEAPFP2 is called to obtain the PTE and XPTE addresses (PCBPTE and PCBXPT respectively).	FREE	
If IEAPFP2 is successful (return code = 0), the XPTPCBQ field of the indicated XPTE is examined. If it equals 0, there is no PCB for this page.		
8 A FIX must be purged if:		
It is a branch entry and the SUSPEND bit is on in the VSL.		
It is an SVC entry that was originally for a FREE (register 0 / 0).		
It is a pseudo branch entry originally for a FREE (that is, entry to FREE was not at FREEX).		
The scan table entry is computed as follows: PVTSCAN + 12(XPTPCBQ - 1)		PURGE
That PCB queue is searched for a PCB whose PCBVBN field matches the virtual address of the page to be freed. If none is found, the scan of the VSL continues. IF PCBTCF of the found PCB = 1, the PCB has no root PCB. If PCBRLP = 0, there is no related PCB.		
For any branch entry, PCBRWK2 of the root PCB pointed to by PCBRTP is compared to register 1. If they are not equal, the root PCB is not for this request. For an SVC entry, the same is true except that register 0 and PCBRWK1 are used in the comparison. If a match is found, the root PCB is marked for interception (for branch of SVC entries only).		
For a pseudo branch entry, if PCBRINT = 1, or if the input TCB address does not match PCBRTCB, or if PCBRWK1 = 0, the search for a related PCB is performed. Otherwise, register 10 is set to a X'30' post code, register 11 to the ECB address from PCBRWK1, register 12 to the TCB address from PCBRTCB, and control is passed to CALLPOST.		
9 If a PCBVBN that matches the virtual address of the page to be freed cannot be found, or PCBFXC = 0 in the found PCB, the VSL scan continues.		TESTQN
Otherwise, processing continues as in Steps 5 and 6 except that fields in the PCB corresponding to those in the PFTE are used, and PFTE queuing is skipped.		
1 CALLPOST sets up parameters for and passes control to POST at entry point IEA0PT01. POST returns control to the caller of CALLPOST.	CALLPOST	

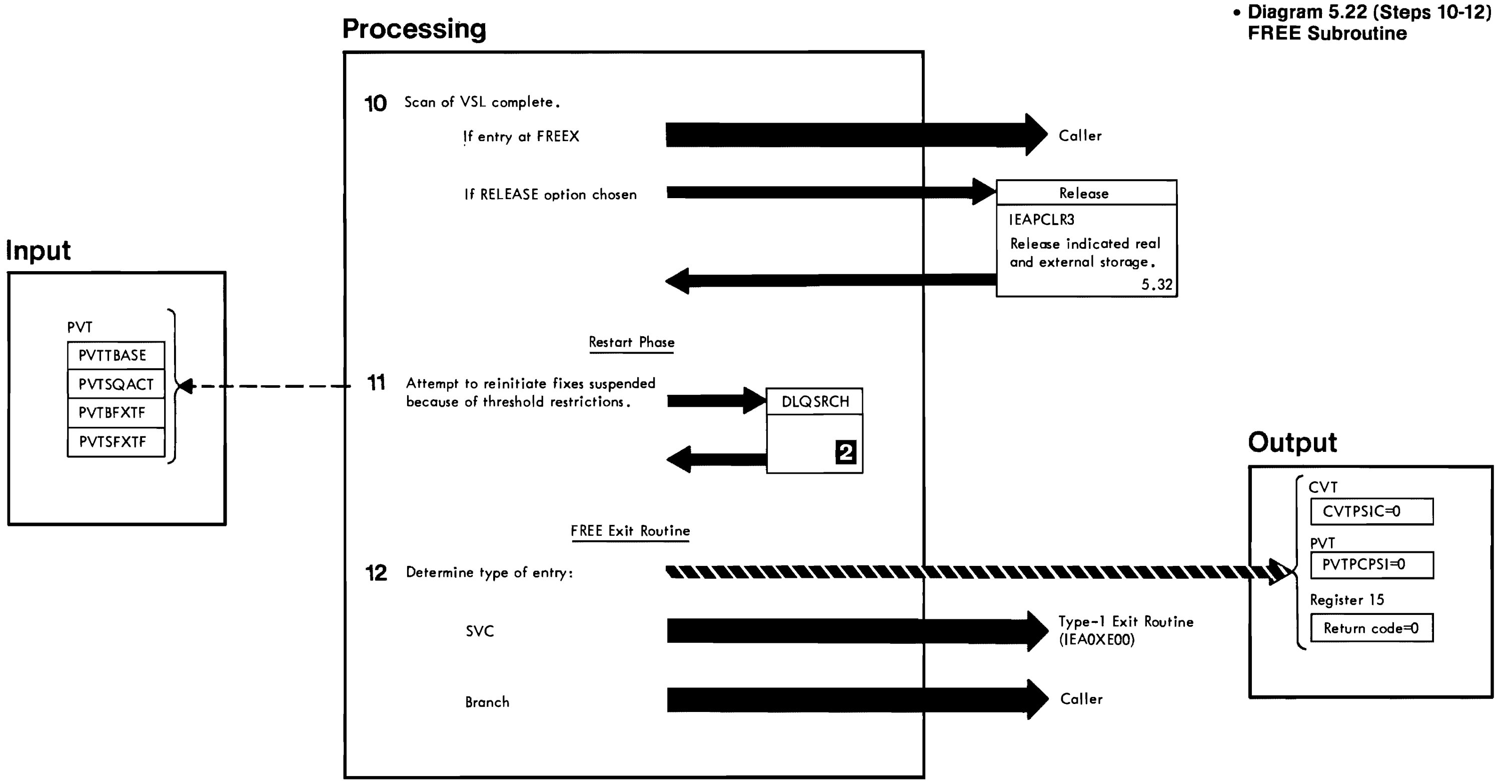

Input
PVT
PVTTBASE
PVTSQACT
PVTBFXTF
PVTSFXTF
Processing
10 Scan of VSL complete.
If entry at FREEX
Caller
If RELEASE option chosen
Release
IEAPCLR3
Release indicated real and external storage.
5.32
Restart Phase
11 Attempt to reinitiate fixes suspended because of threshold restrictions.
DLQSRCH
2
FREE Exit Routine
12 Determine type of entry:
SVC
Type-1 Exit Routine (IEA0XE00)
Branch
Caller
Output
CVT
CVTPSIC=0
PVT
PVTPCPSI=0
Register 15
Return code=0

Diagram 5.22 (Steps 10-12) FREE Subroutine

NOTES	ROUTINE NAME	LABEL
10 For entry at FREEX, internal switches are reset before return.	FREE	ENDLIST
11 DLQSRCH is called with the branch entry FIX delay queue address and the branch entry cutoff threshold (PVTTBASE - PVTSQACT - PVTBFXTF) as parameters. If the return code from DLQSRCH is zero, the SVC entry FIX delay queue must also be searched. Its address and the SVC entry cutoff threshold (PVTTBASE - PVTSQACT - PVTSFXTF) are passed to DLQSRCH. Upon return, or if the first return code from DLQSRCH was 4, processing continues.		RESTART
12		COMEX0
2 DLQSRCH initiates delayed FIX processing as follows:	DLQSRCH	
A If the fixed page count (PVTFXC) is <u>not</u> less than the input cutoff threshold, no reinitiation is possible. Control is returned with a return code of 4.		SRCHLOOP
B If the indicated delay queue is empty, the next queue (if one) must be searched. Control is returned with return code = 0.		TSTDLQ
C If too many pages are needed to fulfill the top (smallest) request on the queue (PCBRWK3 > input threshold - PVTFXC), no reinitiation is possible. Return code = 4.		TSTPCBC
D If the root PCB represents an eligible request, it is dequeued. If it is marked for termination (PCBRAB = 1), no attempt is made to restart the request. Instead, if its count of associated requests is zero (PCBRCNT = 0), it is freed via ROOTFREE. The search then continues at "b".		
E If the root PCB can be used, its address is passed to FIXX to perform FIX processing. Upon return, the search continues at "a".		

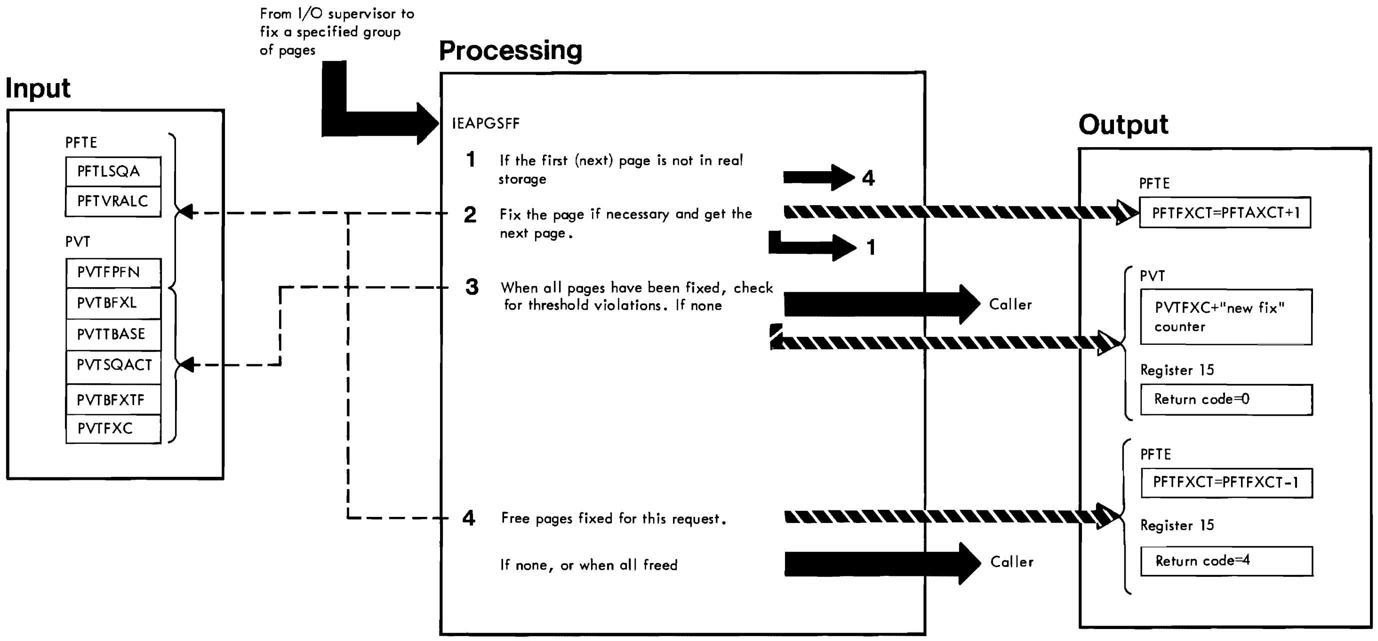
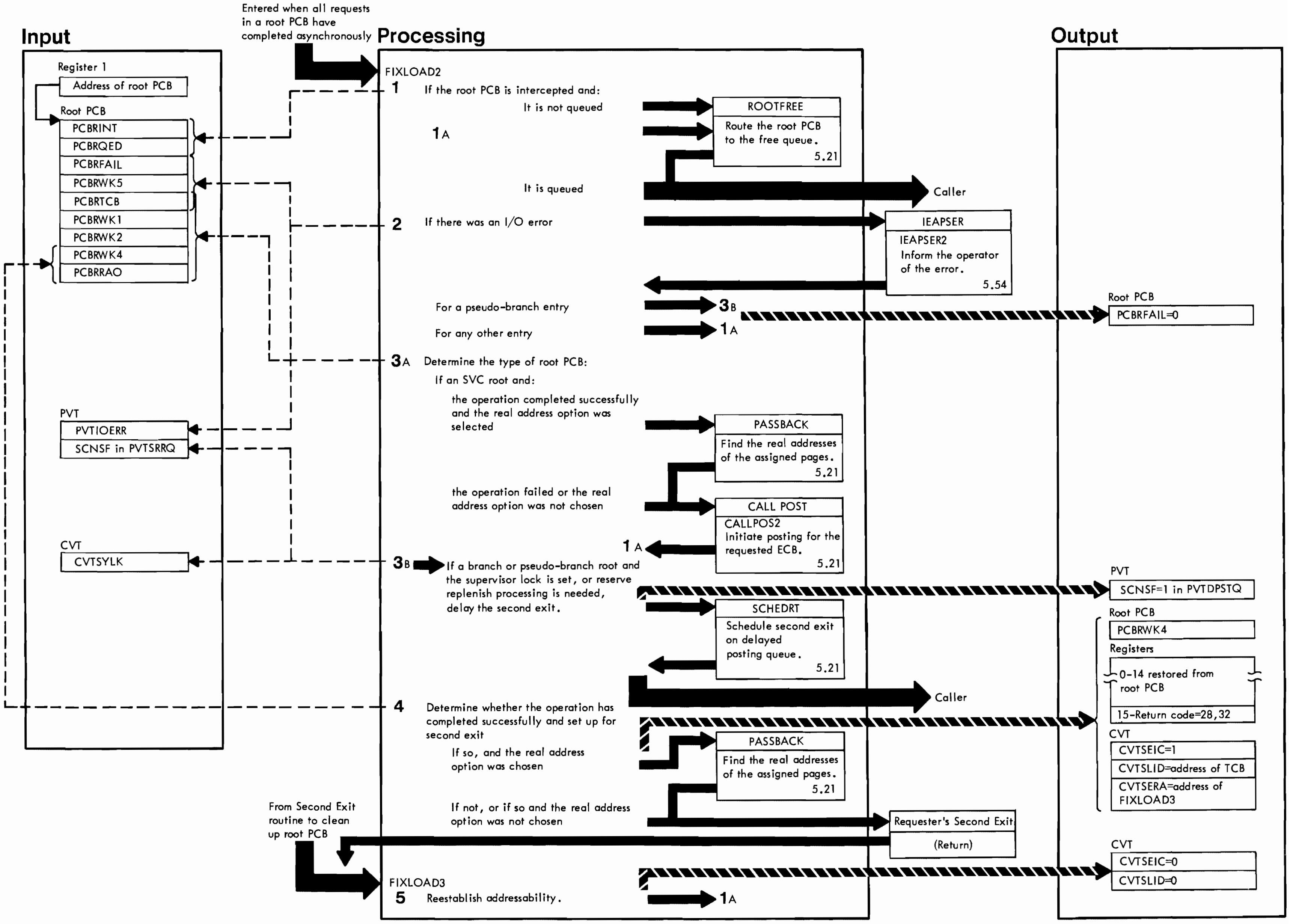

Input
PFTE
PFTLSQA
PFTVRALC
PVT
PVTFPFN
PVTBFXL
PVTTBASE
PVTSQACT
PVTBFXTF
PVTFXC
From I/O supervisor to fix a specified group of pages
Processing
IEAPGSFF
1 If the first (next) page is not in real storage
4
2 Fix the page if necessary and get the next page.
1
3 When all pages have been fixed, check for threshold violations. If none
Caller
4 Free pages fixed for this request.
If none, or when all freed
Caller
Output
PFTE
PFTFXCT=PFTAXCT+1
PVT
PVTFXC+"new fix" counter
Register 15
Return code=0
PFTE
PFTFXCT=PFTFXCT-1
Register 15
Return code=4

Diagram 5.23 Fast FIX Routine (Module IEAPSI)

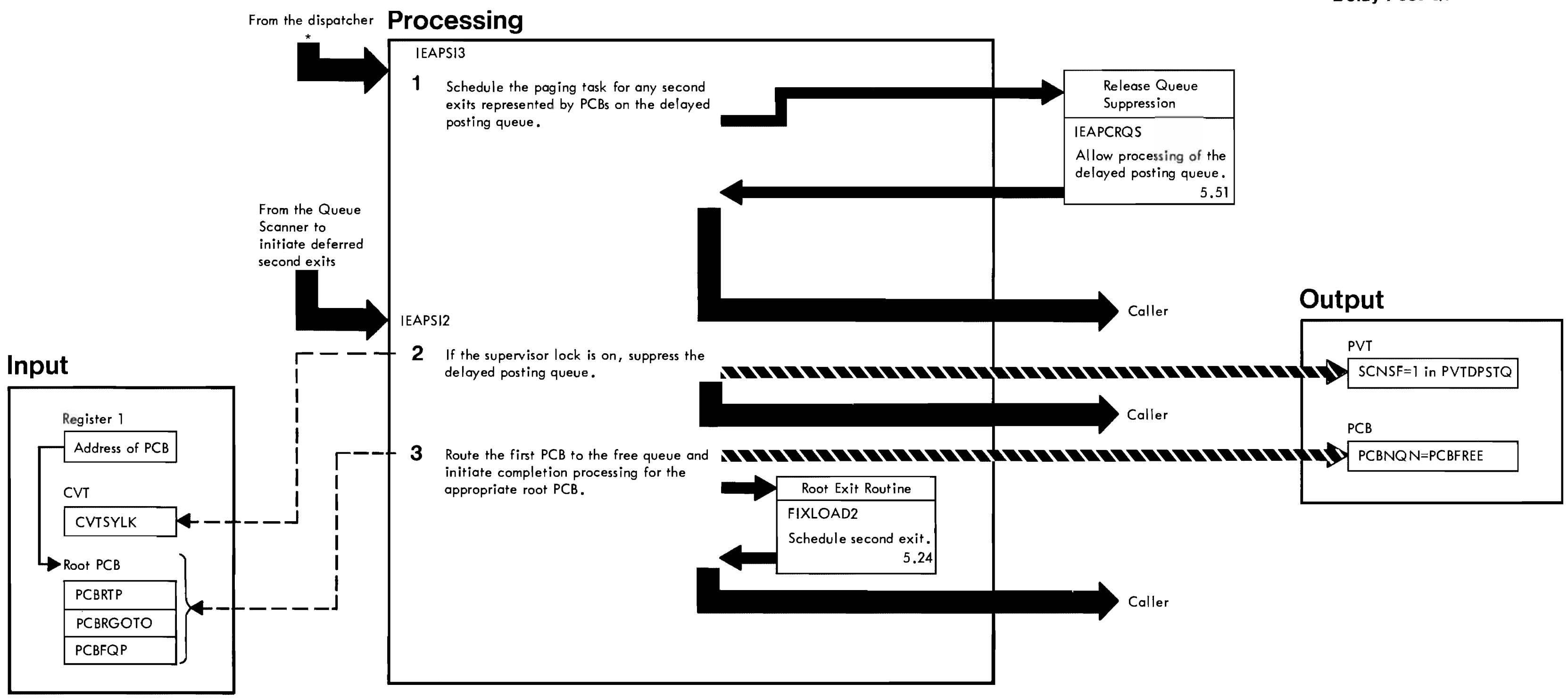
NOTES	ROUTINE NAME	LABEL
1 Internal counters of the number of pages fixed ("fix counter") and the number of newly fixed pages ("new fix counter") are initialized to zero. The test for an in-real-storage page is done with an LRA instruction.	IEAPGSFF	FFBIGLP
2 The real address (obtained from the LRA instruction) is shifted to obtain the PFTE index which is then compared to PVTFPFN. If the PFTE index is less than PFTFPFN, the page is in the nucleus. If it is not, it is determined whether the page is allocated to the SQA, LSQA, or V=R. If any of the tests is positive, the page does not have to be fixed. The page is fixed (PFTFXCT = PFTFXCT + 1) and the "internal fix counter" is incremented. If PFTFXCT now equals 1, the "new fix counter" is also incremented.		FFINLOOP
3 A limit or threshold has been violated if: "New fix counter" is greater than the branch entry fix limit (PVTBXFL). The new system fix count (PVTFXC + "new fix counter") is greater than the branch threshold (PVTTBASE - PVTSQACT - PVTBFXTF).		
4 The same tests as in Step 2 are made to determine whether the page has been fixed. If so, it is freed (PFTFXCT = PFTFXCT - 1) and the "fix counter" is decremented. The continues until the "fix counter" equals zero.		FFBACKUP


Input
Register 1
Address of root PCB
Root PCB
PCBRINT
PCBRQED
PCBRFAIL
PCBRWK5
PCBRTCB
PCBRWK1
PCBRWK2
PCBRWK4
PCBRRAO
PVT
PVTIOERR
SCNSF in PVTSRRQ
CVT
CVTSYLK
Entered when all requests in a root PCB have completed asynchronously
Processing
FIXLOAD2
1 If the root PCB is intercepted and:
It is not queued
1A
It is queued
2 If there was an I/O error
For a pseudo-branch entry
For any other entry
3A Determine the type of root PCB:
If an SVC root and:
the operation completed successfully and the real address option was selected
the operation failed or the real address option was not chosen
3B If a branch or pseudo-branch root and the supervisor lock is set, or reserve replenish processing is needed, delay the second exit.
4 Determine whether the operation has completed successfully and set up for second exit
If so, and the real address option was chosen
If not, or if so and the real address option was not chosen
From Second Exit routine to clean up root PCB
FIXLOAD3
5 Reestablish addressability.
ROOTFREE
Route the root PCB to the free queue.
5.21
Caller
IEAPSER
IEAPSER2
Inform the operator of the error.
5.54
3B
1A
PASSBACK
Find the real addresses of the assigned pages.
5.21
CALL POST
CALLPOS2
Initiate posting for the requested ECB.
5.21
1A
SCHEDRT
Schedule second exit on delayed posting queue.
5.21
Caller
PASSBACK
Find the real addresses of the assigned pages.
5.21
Requester's Second Exit
(Return)
1A
Output
Root PCB
PCBRFAIL=0
PVT
SCNSF=1 in PVTDPSTQ
Root PCB
PCBRWK4
Registers
0-14 restored from root PCB
15-Return code=28,32
CVT
CVTSEIC=1
CVTSLID=address of TCB
CVTSERA=address of FIXLOAD3
CVT
CVTSEIC=0
CVTSLID=0

• Diagram 5.24 FIX/LOAD Asynchronous Completion Routine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
1	FIXLOAD2	
2 The parameters passed to IEAPSER are: Register 0 -- Error code (from PVTIOERR) Register 1 -- Requester's TCB address (from PCBRTCB) If the high-order byte of PCBRWK5 = X'80', the root PCB is for a pseudo branch entry.		
3 If there is a nonzero value in PCBRWK1 (that is, if PCBRWK1 points to an ECB), the root PCB is for an SVC entry.		
If the supervisor lock is on (CVTSYLK = 0), completion processing must be deferred until a pending disabled page fault is satisfied.		FIXLOAD2
If the reserve replenish queue is <u>not</u> suppressed, the pages reserved for SQA allocation must be replenished before completion processing can continue.		
4 The "page supervisor in control" bits (PVTPGMCK) are saved and all are set to zero before the second exit. They are restored upon return. Control is passed to the second exit address via BALR 14,14. Return is to FIXLOAD3 whose address is in CVTSERA.		
5	FIXLOAD3	

Diagram 5.25
Delay Post Queue Handler



Processing

From the dispatcher *

IEAPSI3

1 Schedule the paging task for any second exits represented by PCBs on the delayed posting queue.

From the Queue Scanner to initiate deferred second exits

IEAPSI2

2 If the supervisor lock is on, suppress the delayed posting queue.

3 Route the first PCB to the free queue and initiate completion processing for the appropriate root PCB.

Release Queue Suppression

IEAPCRQS

Allow processing of the delayed posting queue.
5.51

Root Exit Routine

FIXLOAD2

Schedule second exit.
5.24

Caller

Caller

Caller

Input

Register 1

Address of PCB

CVT

CVTSYLK

Root PCB

PCBRTP

PCBRGOTO

PCBFQP

Output

PVT

SCNSF=1 in PVTDPSTQ

PCB

PCBNQN=PCBFREE

• Diagram 5.25 Delay Post Queue Handler (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
* When a second exit cannot be immediately taken after the asynchronous completion of a FIX or LOAD operation for any reason (for example, the supervisor lock is set for a disabled page fault, or a suspended FIX is completed by a restart from FREE), the second exit is deferred and a PCB representing it is placed on the delayed post queue (by FIXLOAD2, the Root Exit routine). IEAPSI3 is invoked by the dispatcher, when the supervisor lock is reset, to initiate second exit processing.	IEAPSI3	
1 When the paging task is dispatched, the Queue Scanner will invoke IEAPSI2 (below) to process the requests on this queue. Interruptions are disabled after entry.		
2 If the supervisor lock has been reset (CVTSYIK = 1), a disabled page fault is pending and the second exit requests must remain deferred.	IEAPSI2	
3 The root PCB is pointed to by PCBRTP and the entry point of the Root Exit routine (FIXLOAD2) is in PCBRGOTO.		
If more than one PCB is on the delayed post queue, the Queue Scanner will reenter IEAPSI2.		
Interruptions are enabled before exit.		

Diagram 5.26
FOE Merge Routine

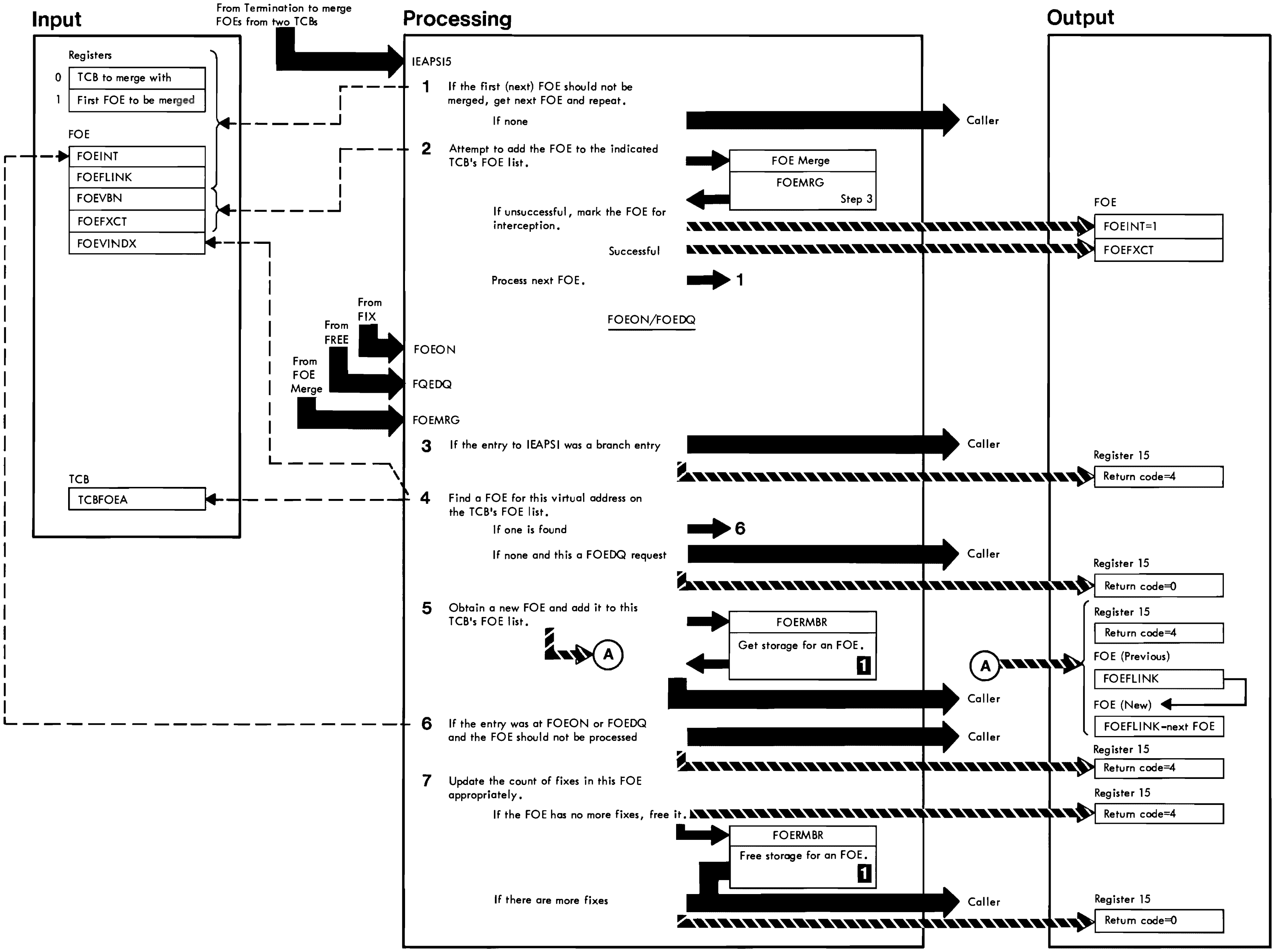
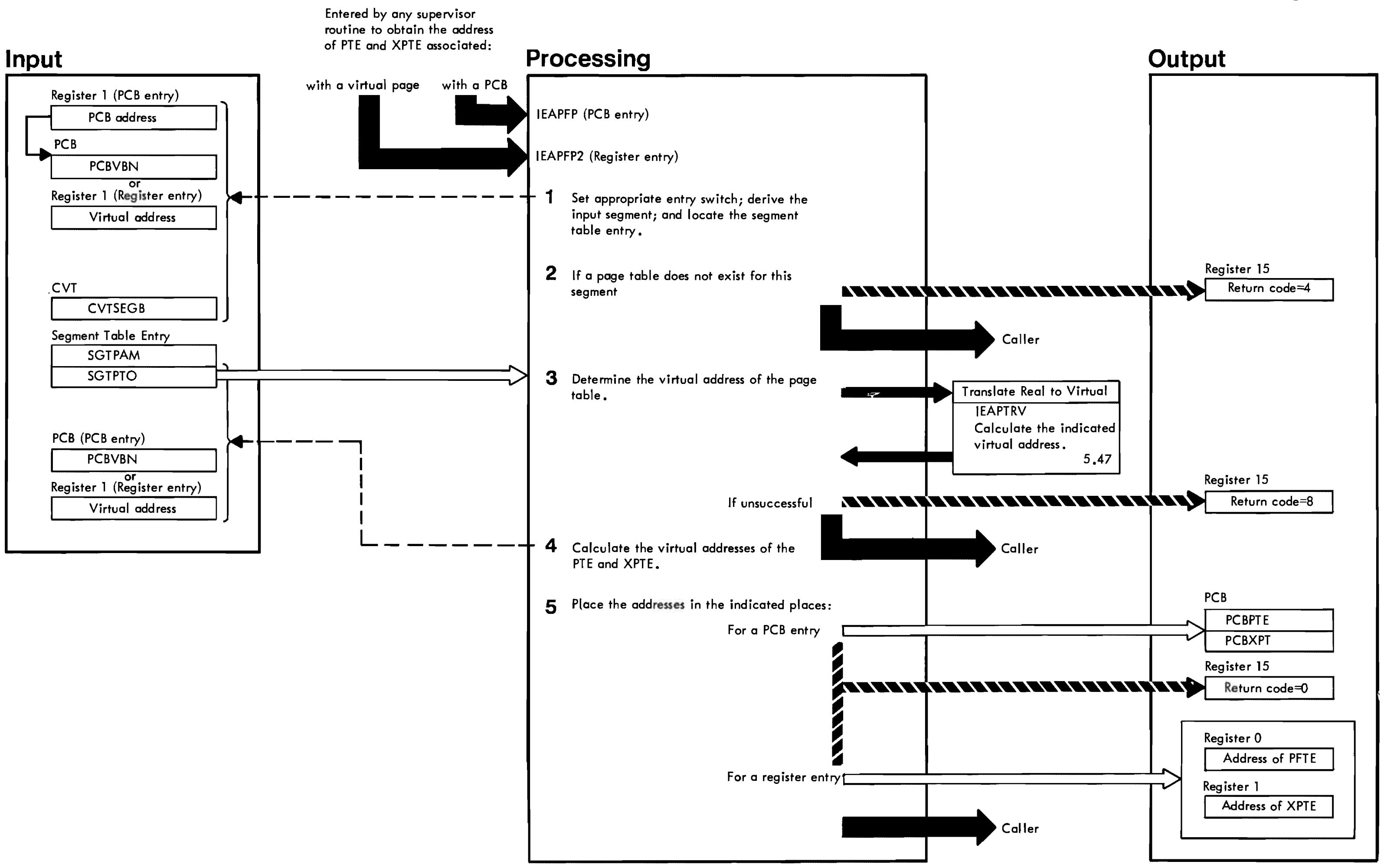


Diagram 5.26 FOE Merge Routine (Module IEAPSI)

NOTES	ROUTINE NAME	LABEL
1 FOEs are chained by the FOEFLINK field. The virtual address (from FOEVBN) and the fix count (from FOEFXCT) are passed to FOEMRG.	IEAPSI5	MERGE1
3 FOEs only exist for SVC entries to IEAPSI.	FOEDQ (all entry points)	FOECM1
4 The input virtual address is compared to FOEVINDX. If they are equal, the FOE has been found. If FOEVINDX is greater than the input virtual address, the search stops, since FOEs are chained in ascending order by FOEVINDX.		FOECM2
5 The address of the next FOE on the chain is placed in the new FOE's FOEFLINK field, and the address of the new FOE is placed in the previous FOE's FOEFLINK. The fix count (FOEFXCT) of the new FOE is set to 1 for entry at FOEON or to the input fix count for entry at FOEMRG.		FOEON2
6 If the FOE is marked for interception (FOEINT = 1), processing on it should not continue.		FOECM3
7 For entry at FOEON, FOEFXCT = FOEFXCT + 1); for entry at FOEDQ, -1. For entry at FOEMRG, the input fix count is added to FOEFXCT. If FOEFXCT now equals zero, the FOE is dequeued (by modifying appropriate FOEFLINK fields) and freed.		FOECM3A
1 FOERMBR obtains (via GETMAIN RMBRANCH) or frees (via FREEMAIN RMBRANCH) storage for an FOE.	FCERMBR	

Diagram 5.27
Find Page Routine



1 Set appropriate entry switch; derive the input segment; and locate the segment table entry.

2 If a page table does not exist for this segment

3 Determine the virtual address of the page table.

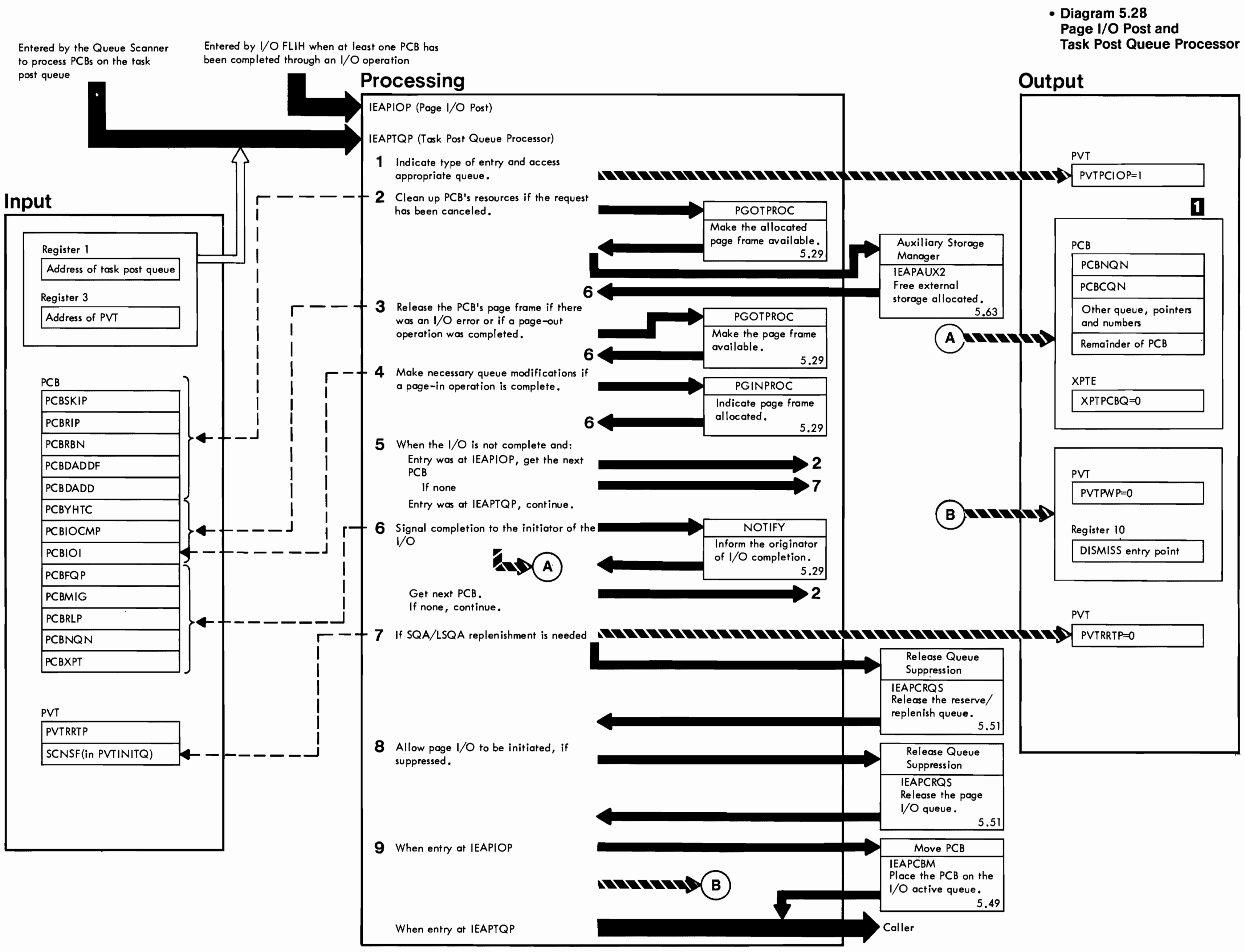
4 Calculate the virtual addresses of the PTE and XPTE.

5 Place the addresses in the indicated places:

Diagram 5.27 Find Page Routine (Module IEAPFP)

NOTES	ROUTINE NAME	LABEL
1 The input segment number equals:		
The high-order byte of PCBVBN for a PCB entry.	IEAPFP	
Byte 1 of register 1 for a Register entry.	IEAPFP2	
The segment table entry is located as follows: CVTSEGB + 4 * input segment number. CVTSEGB is the segment table origin. 4 is the length of an STE.	IEAPFP and IEAPFP2	FINDPG10
2 If SGTPAM=1, no page table exists for this STE.		
3 The real storage address of the page table origin (PTO) is in SGTPTO.		FINDPG20
4 The virtual address of the PTE = virtual address (PTO) + L*P. L is the length of a PTE (2 bytes). P is the page number (bits 8-11 of the PCBVBN or bits 16-19 of register 1).		FINDPG30
The virtual address of the XPTE = virtual address (PTO) + 32 + L*P L is the length of an XPTE (8 bytes). P is the page number.		

• **Diagram 5.28**
Page I/O Post and
Task Post Queue Processor



Entered by the Queue Scanner to process PCBs on the task post queue
Entered by I/O FLIH when at least one PCB has been completed through an I/O operation
Input
Register 1
Address of task post queue
Register 3
Address of PVT
PCB
PCBSKIP
PCBRIP
PCBRBN
PCBDADDF
PCBDADD
PCBYHTC
PCBIOCMP
PCBIOI
PCBFQP
PCBMIG
PCBRLP
PCBNQN
PCBXPT
PVT
PVTRRTP
SCNSF(in PVTINITQ)
Processing
PGOTPROC
Make the allocated page frame available.
5.29
Auxiliary Storage Manager
IEAPAUX2
Free external storage allocated.
5.63
PGOTPROC
Make the page frame available.
5.29
PGINPROC
Indicate page frame allocated.
5.29
NOTIFY
Inform the originator of I/O completion.
5.29
Release Queue Suppression
IEAPCRQS
Release the reserve/replenish queue.
5.51
Release Queue Suppression
IEAPCRQS
Release the page I/O queue.
5.51
Move PCB
IEAPCBM
Place the PCB on the I/O active queue.
5.49
Caller
6
6
6
2
7
2
A
A
B
B
Output
PVT
PVTPCIOP=1
1
PCB
PCBNQN
PCBCQN
Other queue, pointers and numbers
Remainder of PCB
XPTE
XPTPCBQ=0
PVT
PVTPWP=0
Register 10
DISMISS entry point
PVT
PVTRRTP=0

IEAPIOP (Page I/O Post)

IEAPTQP (Task Post Queue Processor)

1 Indicate type of entry and access appropriate queue.

2 Clean up PCB's resources if the request has been canceled.

3 Release the PCB's page frame if there was an I/O error or if a page-out operation was completed.

4 Make necessary queue modifications if a page-in operation is complete.

5 When the I/O is not complete and:
Entry was at IEAPIOP, get the next PCB
If none
Entry was at IEAPTQP, continue.

6 Signal completion to the initiator of the I/O

Get next PCB.
If none, continue.

7 If SQA/LSQA replenishment is needed

8 Allow page I/O to be initiated, if suppressed.

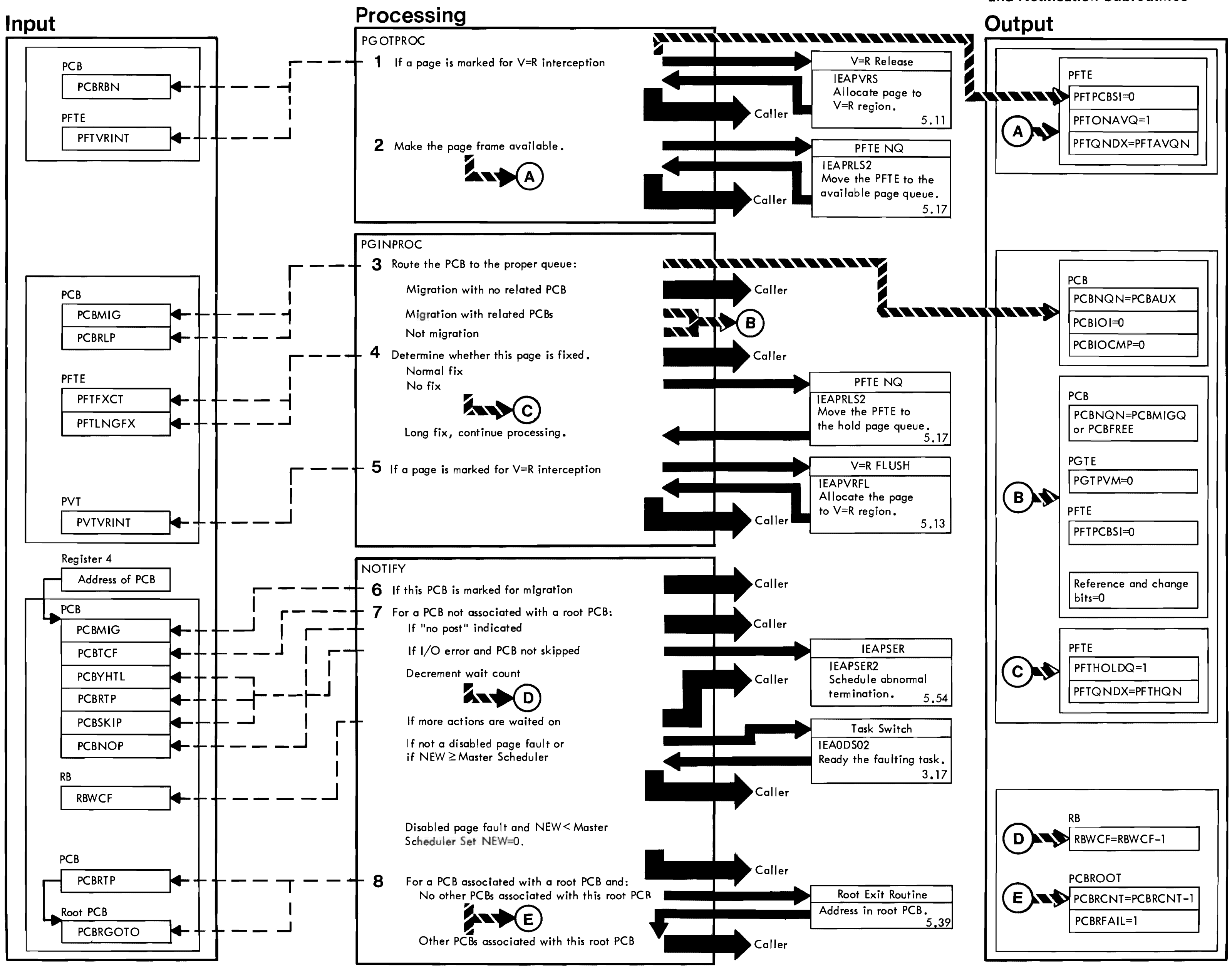
9 When entry at IEAPIOP

When entry at IEAPTQP

• Diagram 5.28 Page I/O Post and Task Post Queue Processor (Module IEAPIOP)

NOTES		ROUTINE NAME	LABEL
1	PVTPCIOP is turned on at entry and off at exit. A switch is set to indicate entry at IEAPTQP.	IEAPTQP and IEAPIOP	IEAPTQP IEAPIOP
2	This processing is attempted whenever the PCBSKIP flag is on. If the "release-in-post" flag (PCBRIP) is off, go immediately to Step 6. PGOTPROC is called only if PCBRBN=0, and IEAPAUX2 is called only if PCBDADDF=1.		COMMON
3			CANCEL
4			TESTYPE
5			NOERR
6	If the PCB has related PCBs, this "notification" logic is performed for each PCB on the related chain.		GONOTIF1
7	PVTRRTP is set by the Reserve Replenish Queue Processor when it has more work to do but there are PCBs on the task post queue or by Page Replacement when no pages could be replaced. If PVTRRTP=1, it signals the Task Post Queue Processor to release the reserve replenish queue.		
8			ENDPROCX
9			NOTSURP
1	The next queue number (PCBNQN) is set to the free queue unless the PCB is marked for migration, in which case PCBNQN is set to the migration queue. If the PCB has related PCBs and: • Entry was at IEAPTQP, PCBCQN in the related PCBs is set to the task post queue. • Entry was at IEAPIOP, PCBCQN in the related PCBs is set to the I/O active queue. If the PCB has no related PCBs or when all related PCBs have been handled, and if the original PCB was for migration, the original PCB is reinitialized by setting all fields other than queue pointers and queue numbers (PCBFQP, PCBBQP, PCBNQN, PCBCQN) to zero. In addition, if PCBXPT≠0 in the original PCB, XPTPCBQ of the XPTE pointed to by PCBXPT is set to zero.		LVXPT

• **Diagram 5.29**
Page I/O Post—Page-out, Page-in, and Notification Subroutines



PGOTPROC

1 If a page is marked for V=R interception

2 Make the page frame available.
A

PGINPROC

3 Route the PCB to the proper queue:
Migration with no related PCB
Migration with related PCBs
B
Not migration

4 Determine whether this page is fixed.
Normal fix
No fix
C
Long fix, continue processing.

5 If a page is marked for V=R interception

NOTIFY

6 If this PCB is marked for migration

7 For a PCB not associated with a root PCB:
If "no post" indicated
If I/O error and PCB not skipped
Decrement wait count
D
If more actions are waited on
If not a disabled page fault or if NEW ≥ Master Scheduler

Disabled page fault and NEW < Master Scheduler Set NEW=0.

8 For a PCB associated with a root PCB and:
No other PCBs associated with this root PCB
E
Other PCBs associated with this root PCB

• Diagram 5.29 Page I/O Post - Page-out, Page-in, and Notification Subroutines (Module IEAPIOP)

NOTES	ROUTINE NAME	LABEL
1	PGOTPROC	PGOTPROC
2 If the page-out was for a swap-out, the PFTE is added to the available queue, since it cannot be reclaimed.		
3 A PCB for migration with no related PCBs is routed to the auxiliary storage allocation queue and marked for page-out (PCBIOI=0) for continuation of the migration process for this page. A PCB for migration with related PCBs is routed to the migration queue so that migration can continue with a different page.	PGINPROC	PGINPROC
5 V=R Flush is called to move the page above the V=R line (if not long-fixed) or to notify V=R of a long-fix.		CHECKVR
6	NOTIFY	NOTIFY
7 If the page represents a satisfied page fault (PCBYHTC=0 and PCBNOP=0) and the faulting task is waiting for this event only (REWCF-1=0), then Task Switch is called, unless for a disabled page fault task and NEW < Master Scheduler.		
8 If the original PCB indicates I/O error (PCBYHTC=1), then NOTIFY indicates this in the root PCB (PCBRFAIL=1).		

Diagram 5.30
Release Routine (User SVC)
(Releasing Real and External Page Storage)

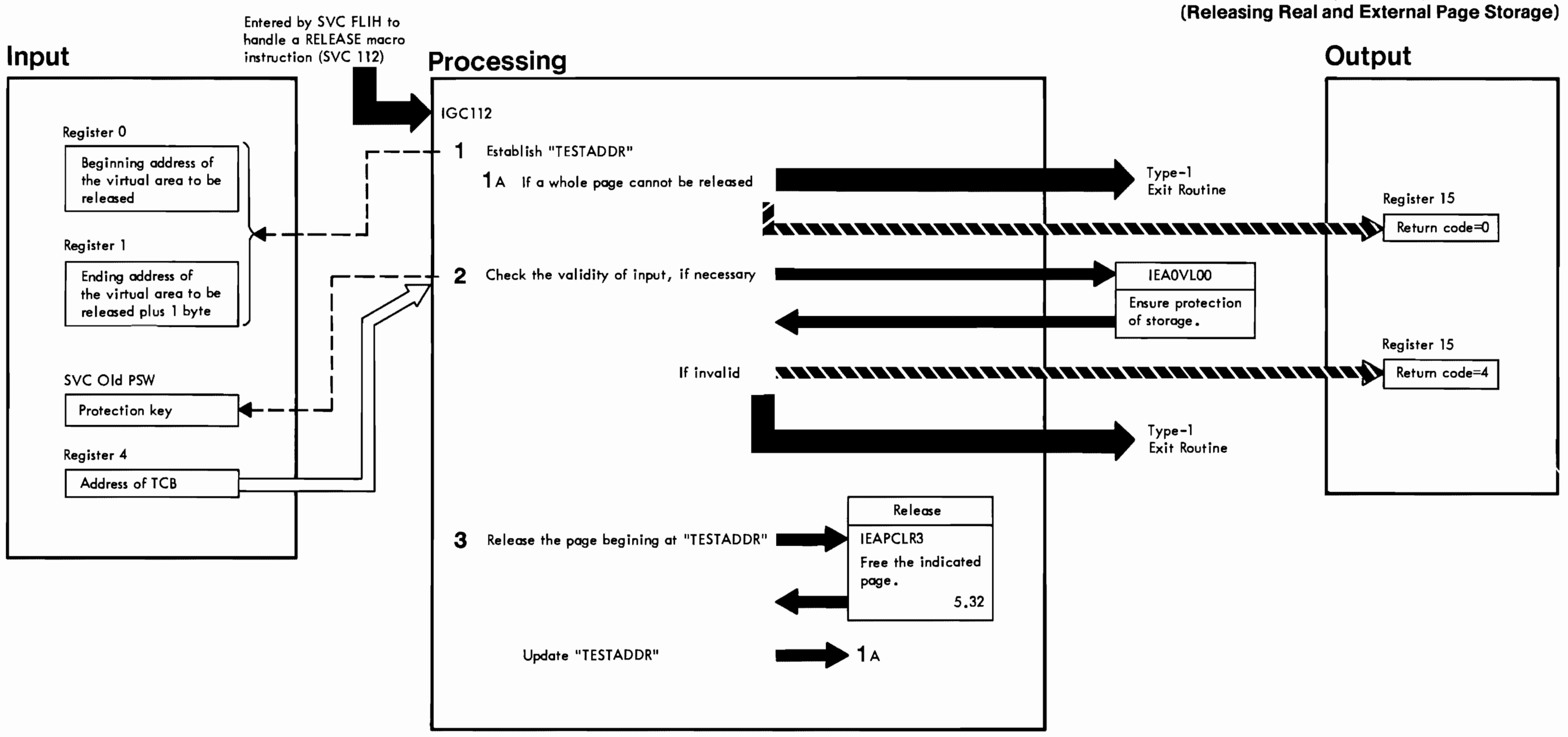
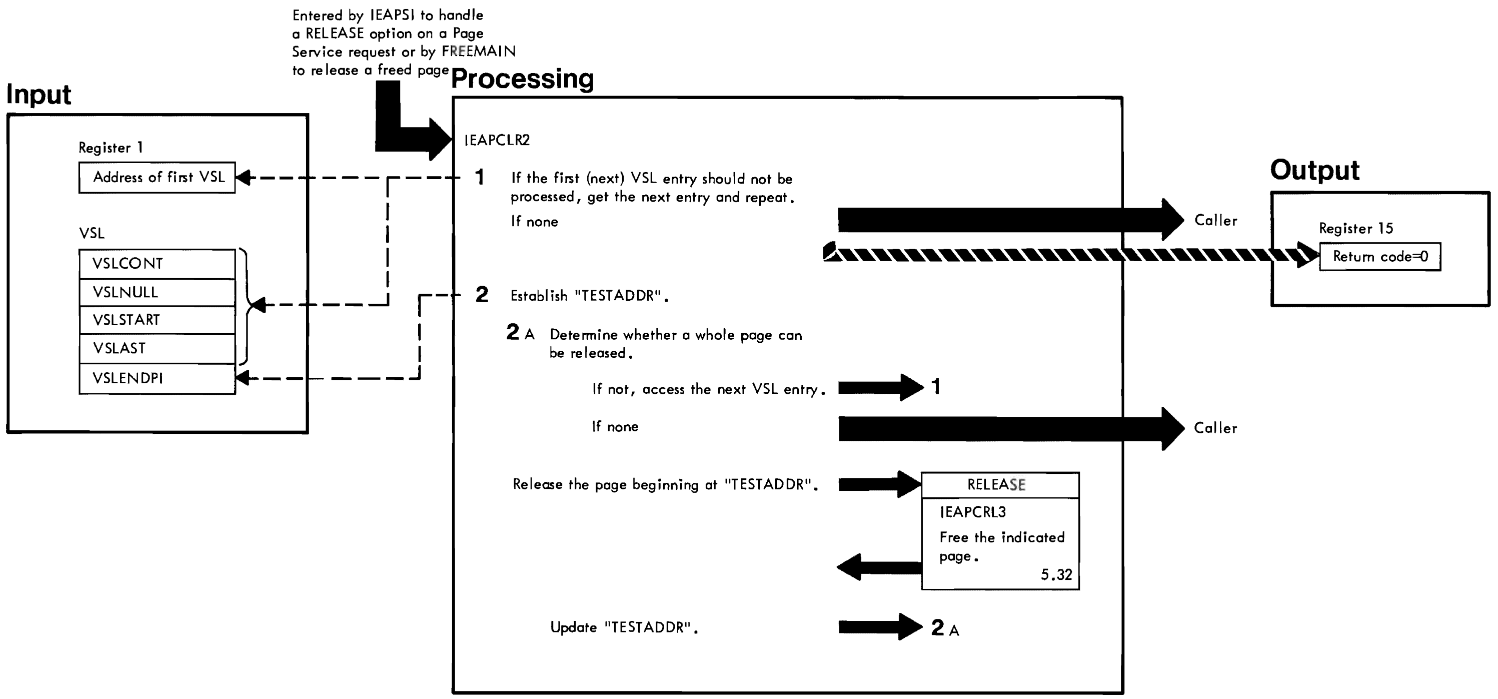

Input
Register 0
Beginning address of the virtual area to be released
Register 1
Ending address of the virtual area to be released plus 1 byte
SVC Old PSW
Protection key
Register 4
Address of TCB
Entered by SVC FLIH to handle a RELEASE macro instruction (SVC 112)
Processing
IGC112
1 Establish "TESTADDR"
1A If a whole page cannot be released
Type-1 Exit Routine
2 Check the validity of input, if necessary
IEAOVL00
Ensure protection of storage.
If invalid
Type-1 Exit Routine
3 Release the page begining at "TESTADDR"
Release
IEAPCLR3
Free the indicated page.
5.32
Update "TESTADDR"
1A
Output
Register 15
Return code=0
Register 15
Return code=4

Diagram 5.30 Release Routine (User SVC) (Module IEAPCLR)

NOTES	ROUTINE NAME	LABEL
1 The paging-supervisor-in-control flag (CVTPGSIC) is turned on at entry and off at exit.	IGC112	
2 TESTADDR is an internal pointer. Initially, it is set to the value in register 0 rounded up to the next higher page boundary. Each time a page is released it is incremented by 4096.		
If the input in register 1 minus TESTADDR is less than 4096, a whole page cannot be released.		
3 If the caller does not have a protection key of 0, the validity of the address must be checked to ensure that the caller is authorized to release this page.		
On entry to IEAPCLR3, register 1 = TESTADDR and register 3 = PVT address.		

• **Diagram 5.31**
Release Routine (Supervisor Branch)
(Releasing Real and External Page Storage)


Entered by IEAPSI to handle a RELEASE option on a Page Service request or by FREEMAIN to release a freed page
Input
Register 1
Address of first VSL
VSL
VSLCONT
VSLNULL
VSLSTART
VSLAST
VSLENDPI
Processing
IEAPCLR2
1 If the first (next) VSL entry should not be processed, get the next entry and repeat.
If none
Caller
2 Establish "TESTADDR".
2 A Determine whether a whole page can be released.
If not, access the next VSL entry.
1
If none
Caller
Release the page beginning at "TESTADDR".
RELEASE
IEAPCRL3
Free the indicated page.
5.32
Update "TESTADDR".
2 A
Output
Register 15
Return code=0

• Diagram 5.31 Release Routine (Supervisor Branch) (Module IEAPCLR)

NOTES	ROUTINE NAME	LABEL
1 PVTPCCLR is turned on at entry and off at exit. If VSLCONT=1, this VSL is not to be processed and the VSL pointed to by VSLSTART is accessed next. If VSLNULL=1, this VSL is skipped and the next entry is accessed. If VSLAST=1, this is the last VSL.	IEAPCLR2	
2 TESTADDR is handled as in IGC112, Note 2, except that it is initially set to the value in VSLSTART of the VSL being processed and is subtracted from VSLENDP1 to determine if a whole page can be released.		
3 On entry to IEAPCLR3, register 1 = TESTADDR and register 3 = PVT address.		

Diagram 5.32 (Steps 1-6)
Release Routine (Branch)
(Releasing Real and External Page Storage)

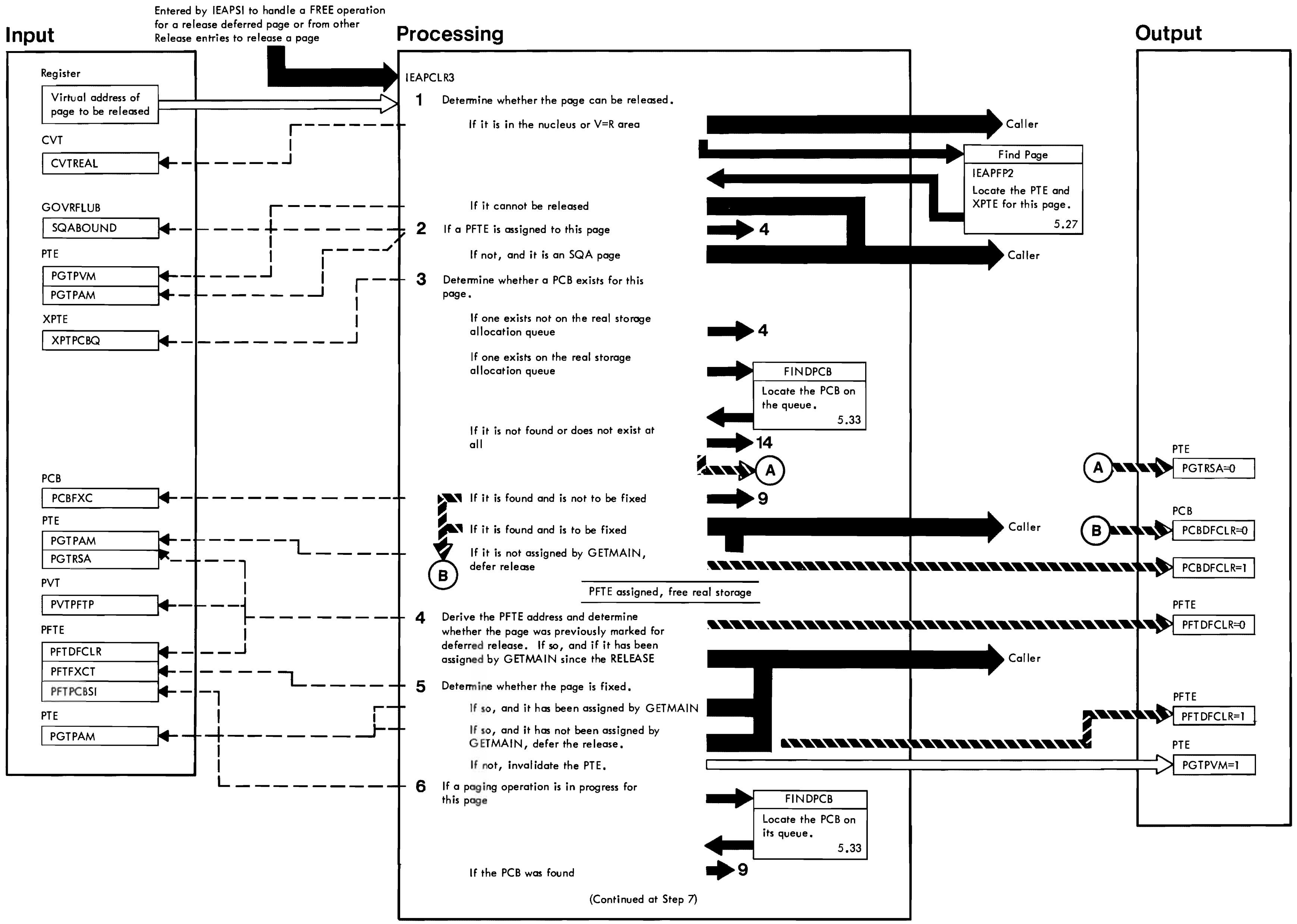
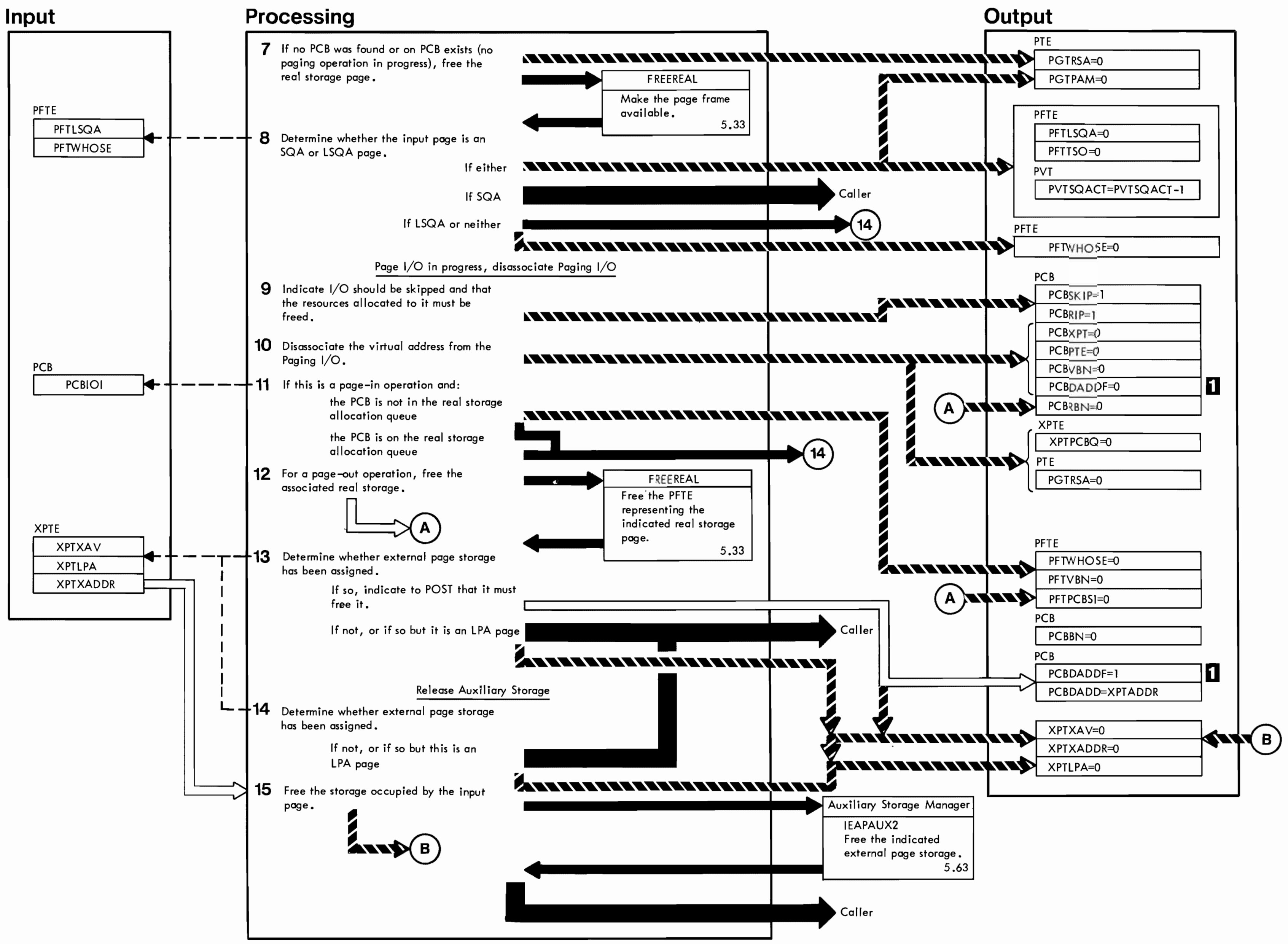

Input
Entered by IEAPSI to handle a FREE operation for a release deferred page or from other Release entries to release a page
Processing
Output
Register
Virtual address of page to be released
CVT
CVTREAL
GOVRFLUB
SQABOUND
PTE
PGTPVM
PGTPAM
XPTE
XPTPCBQ
PCB
PCBFXC
PTE
PGTPAM
PGTRSA
PVT
PVTPFTP
PFTE
PFTDFCLR
PFTFXCT
PFTPCBSI
PTE
PGTPAM
IEAPCLR3
1 Determine whether the page can be released.
If it is in the nucleus or V=R area
Caller
Find Page
IEAPFP2
Locate the PTE and XPTE for this page.
5.27
If it cannot be released
2 If a PFTE is assigned to this page
4
If not, and it is an SQA page
Caller
3 Determine whether a PCB exists for this page.
If one exists not on the real storage allocation queue
4
If one exists on the real storage allocation queue
FINDPCB
Locate the PCB on the queue.
5.33
If it is not found or does not exist at all
14
A
If it is found and is not to be fixed
9
If it is found and is to be fixed
Caller
If it is not assigned by GETMAIN, defer release
B
PFTE assigned, free real storage
4 Derive the PFTE address and determine whether the page was previously marked for deferred release. If so, and if it has been assigned by GETMAIN since the RELEASE
Caller
5 Determine whether the page is fixed.
If so, and it has been assigned by GETMAIN
If so, and it has not been assigned by GETMAIN, defer the release.
If not, invalidate the PTE.
6 If a paging operation is in progress for this page
FINDPCB
Locate the PCB on its queue.
5.33
If the PCB was found
9
(Continued at Step 7)
A
PTE
PGTRSA=0
B
PCB
PCBDFCLR=0
PCBDFCLR=1
PFTE
PFTDFCLR=0
PFTE
PFTDFCLR=1
PTE
PGTPVM=1

Diagram 5.32 (Steps 1-6) Release Routine (Branch) (Module IEAPCLR)

NOTES	ROUTINE NAME	LABEL
1 If CVTREAL is greater than the input address, the page is in the nucleus or V=R area and cannot be released. If the FINDPAGE return code is not zero, the segment is unassigned or an internal error was found. The page cannot be released. If the input address is greater than or equal to SQABOUND, the page is in the SQA area and cannot be released. If the page is invalid (PGTPVM=1), it cannot be released.	IEAPCLR3	
3 If the PCB is found on the real storage allocation queue and it is for a FIX request for a page that is not assigned by GETMAIN the deferred clear flag in the PCB is set (PCBDFCLR=1). When IEAPALOC obtains a PFTE for this page it will set the deferred clear flag in the PFTE. When the fix is removed by FREE, FREE will call IEAPCLR3 to release the page.		
4 The PFTE is located as follows: PFTE index (from PGTRSA) + PVTPFTP (the PFT origin) = PFTE address. If the deferred clear flag is on (PFTDFCLR=1), IEAPCLR3 was probably called by FREE to release a page that could not be released before because it was fixed. If, however, GETMAIN reassigned the page between the time the release was requested and the time FREE removed the fix (if PGTPAM=1), the release is ignored.		
5 If the page is fixed and is not assigned by GETMAIN, the release must be deferred until the fix is undone by FREE. The FREE processor in IEAPSI will call IEAPCLR3 again to release the page. The storage key is set to zero and fetch protection is indicated so that the user cannot have valid access to the page while the release is pending (unless another GETMAIN is issued).		

Diagram 5.32 (Steps 7-15)
Release Routine (Branch)
(Releasing Real and External Page Storage)



7 If no PCB was found or on PCB exists (no paging operation in progress), free the real storage page.

8 Determine whether the input page is an SQA or LSQA page.

Page I/O in progress, disassociate Paging I/O

9 Indicate I/O should be skipped and that the resources allocated to it must be freed.

10 Disassociate the virtual address from the Paging I/O.

11 If this is a page-in operation and:
the PCB is not in the real storage allocation queue
the PCB is on the real storage allocation queue

12 For a page-out operation, free the associated real storage.

13 Determine whether external page storage has been assigned.
If so, indicate to POST that it must free it.
If not, or if so but it is an LPA page

Release Auxiliary Storage

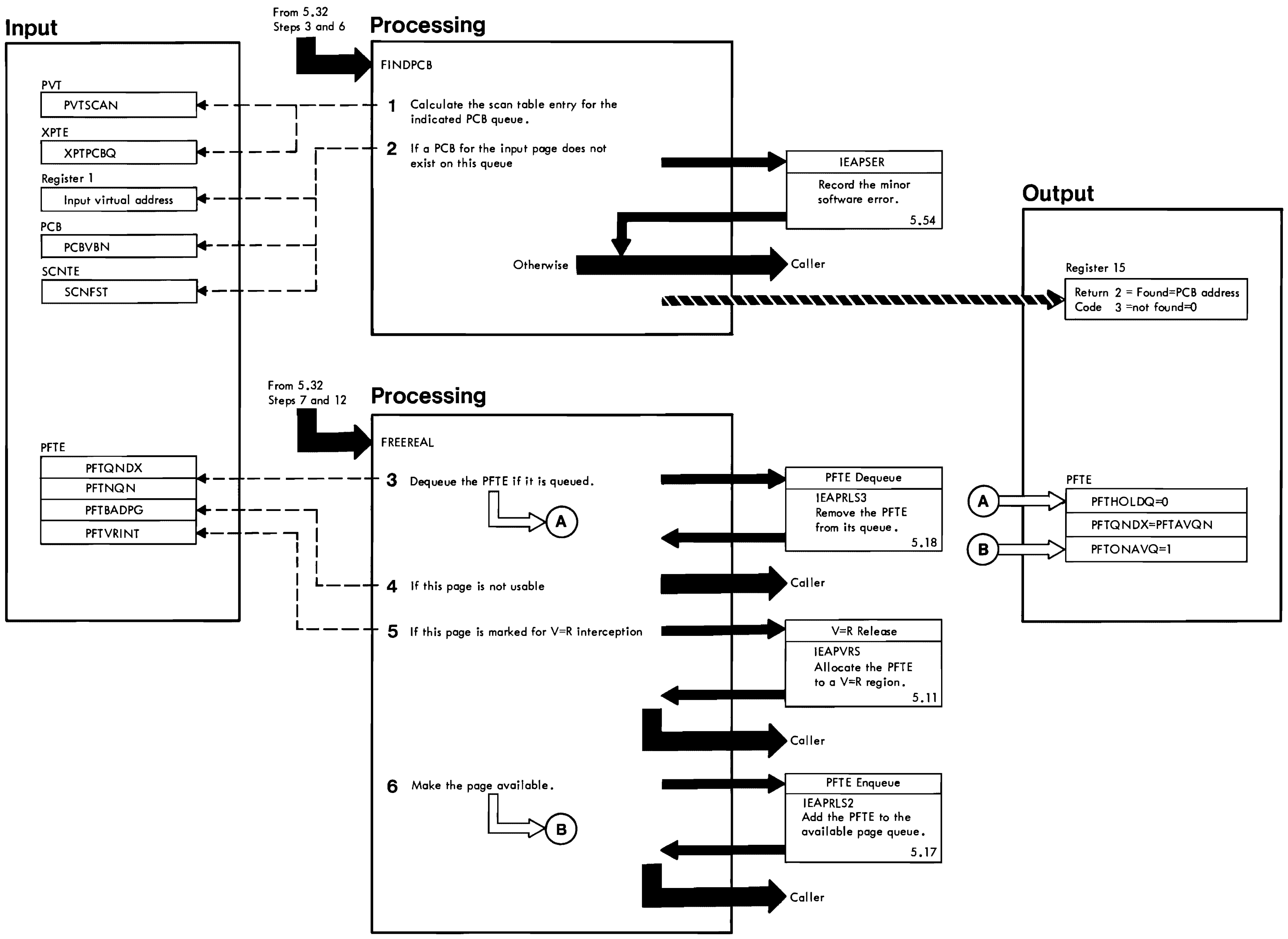
14 Determine whether external page storage has been assigned.
If not, or if so but this is an LPA page

15 Free the storage occupied by the input page.

Diagram 5.32 (Steps 7-15) Release Routine (Branch) (Module IEAPCLR)

NOTES	ROUTINE NAME	LABEL
7 The real storage address (PGTRSA) is set to zero to prevent future page reclamation.	IEAPCLR3	
8 Since SQA pages are not paged and so have no XPTEs assigned to them, the normal route cannot be taken. Therefore, no further action on them need be taken.		
9 Setting PCBRIP=1 informs IEAPIOP (Page I/O Post) that it must free either real or auxiliary storage fcr the page.		
11 For a page-in, the Release routine frees auxiliary storage and Post frees main storage according to PCBRBN.		
12 For a page-out, the Release routine frees main storage and Post.		
13 Frees auxiliary storage if any was assigned.		
14 The only time the XPTLPA flag should be on is when IEAPCLR3 is entered by FREEMAIN after a TSO user has logged off.		
1 PCBDADDF and PCBRBN are tested by IEAPIOP to determine which type of storage it must attempt to free. If PCBDADDF=1, the page I/O routine Post will free auxiliary storage using the encoded address in PCBDADD. If PCBRBN≠0, Post will free real storage using the address in PCBRBN.		

Diagram 5.33
Release—Subroutines for Searching PCBs and Freeing Real Storage



1 Calculate the scan table entry for the indicated PCB queue.

2 If a PCB for the input page does not exist on this queue

3 Dequeue the PFTE if it is queued.

4 If this page is not usable

5 If this page is marked for V=R interception

6 Make the page available.

Diagram 5.33 Release - Subroutines for Searching PCBs and Freeing Real Storage (Module IEAPCLR)

NOTES	ROUTINE NAME	LABEL
1 The scan table entry is located as follows: PVTSCAN + (XPTPCBQ-1)*12.	FINDPCB	
2 The PCB is searched for by comparing bytes 1 and 2 of the input virtual address with PCBVBN. If they are equal, the PCB has been found. If the queue is empty (SCNFST=0) or the search fails, an error condition exists (PFTPCBSI or XPTPCBQ is incorrectly set).		
3 The PFTE is on a queue if PFTQNDX≠PFTNQN. The page is not usable if the "RMS-detected-solid-storage-failure" flag is on (PFTBADPG=1).	FREEREAL	
5 If PFTVRINT=1 (set by IEAPVEQR), the page is needed for a V=R region.		
6 IEAPRLS2 adds the PFTE so that it will be removed first since the released page cannot be reclaimed.		

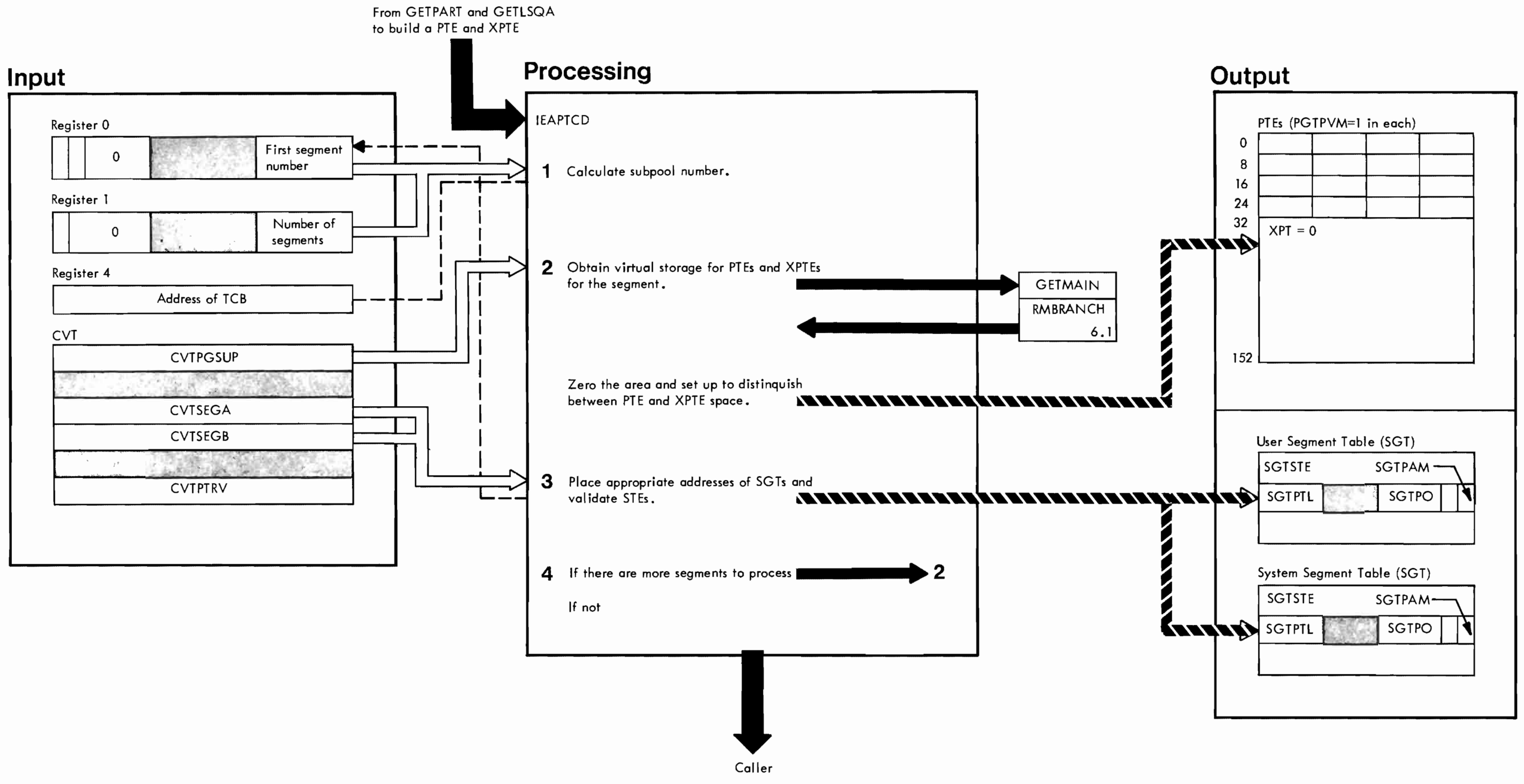
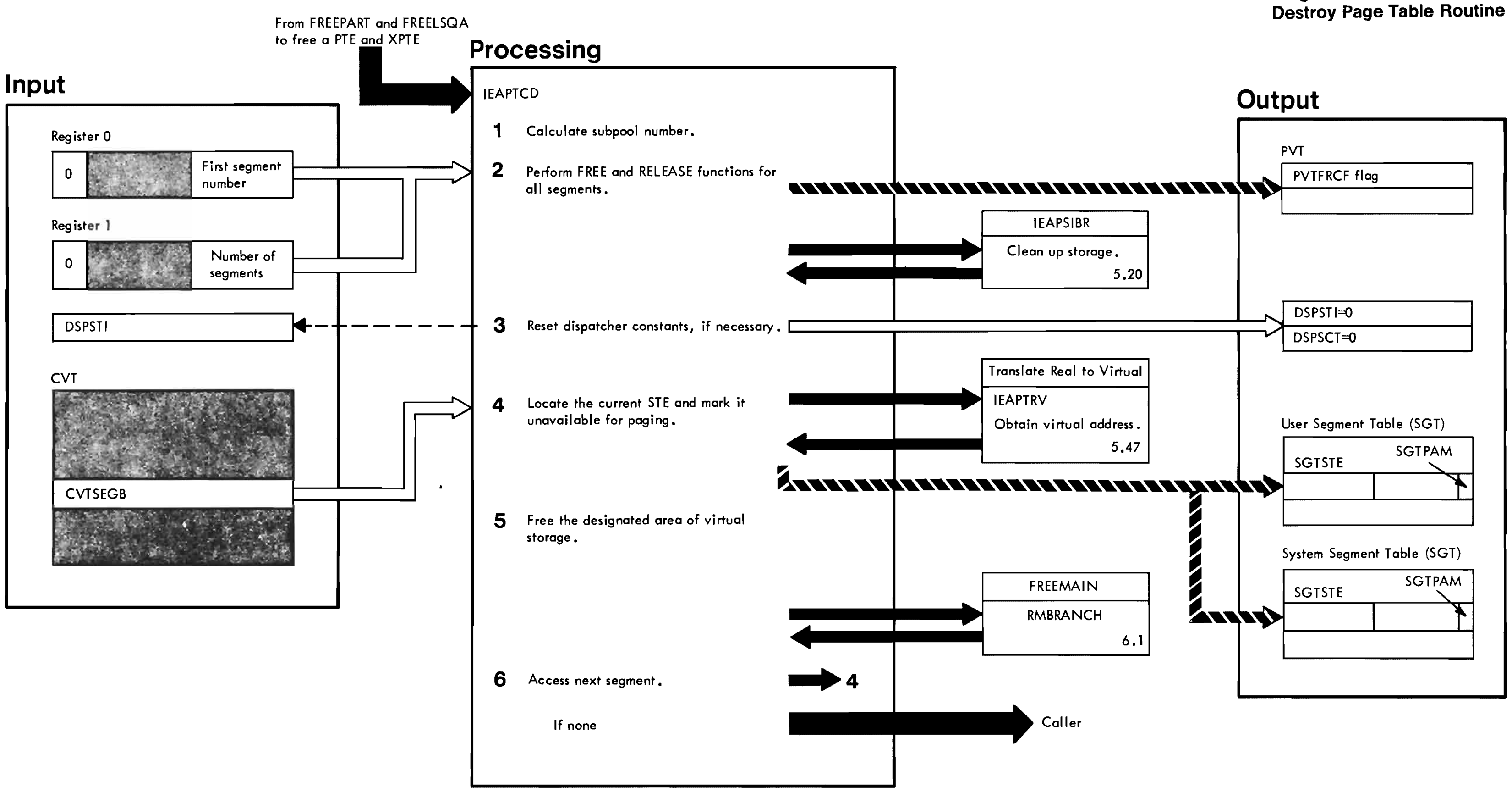

From GETPART and GETLSQA to build a PTE and XPTE
Input
Register 0
0
First segment number
Register 1
0
Number of segments
Register 4
Address of TCB
CVT
CVTPGSUP
CVTSEGA
CVTSEGB
CVTPTRV
Processing
IEAPTCD
1 Calculate subpool number.
2 Obtain virtual storage for PTEs and XPTEs for the segment.
GETMAIN
RMBRANCH
6.1
Zero the area and set up to distinquish between PTE and XPTE space.
3 Place appropriate addresses of SGTs and validate STEs.
4 If there are more segments to process
2
If not
Caller
Output
PTEs (PGTPVM=1 in each)
0
8
16
24
32
XPT = 0
152
User Segment Table (SGT)
SGTSTE
SGTPAM
SGTPTL
SGTPO
System Segment Table (SGT)
SGTSTE
SGTPAM
SGTPTL
SGTPO

Diagram 5.34 Create Page Table Routine (Module IEAPTCD)

NOTES	ROUTINE NAME	LABEL
1 If the "page-table-in-SQA" option is chosen (bit 0 on in register 1), subpool 245 is used. Otherwise, subpool 255 is used.	Create Page Table	IEAPTCD
2 The parameters passed to GETMAIN are: Register 0 - Byte 0 - subpool number Bytes 1-3 - 160 (dec.) Register 1 - any negative value		CREATE00
Upon return from GETMAIN, the entire 160-byte area is zeroed and PGTPVM is set to 1 in each of the first 16 halfwords in the area.		CREATE10
3 The real address of this area is placed in both the system and user SGTs as follows: • The STE displacement = segment-number-being-processed times 4. • This displacement is added to CVTSEGA and CVTSEGB to get the STE addresses. • The real address of the page table (obtained via an LRA instruction using the address returned by GETMAIN as input) is placed in SGTSTE of both STEs. • SGTPAM is set to 0 (validated) in the system STE. • If the validate-UST option is chosen (register 0, byte 0, bit 1 = 1), SGTPAM is set to 0 in the user STE. Otherwise, it is set to 1.		CREATE30

Diagram 5.35
Destroy Page Table Routine



From FREEPART and FREELSQA to free a PTE and XPTE
Input
Register 0
0
First segment number
Register 1
0
Number of segments
DSPSTI
CVT
CVTSEGB
Processing
IEAPTCD
1 Calculate subpool number.
2 Perform FREE and RELEASE functions for all segments.
IEAPSIBR
Clean up storage.
5.20
3 Reset dispatcher constants, if necessary.
Translate Real to Virtual
IEAPTRV
Obtain virtual address.
5.47
4 Locate the current STE and mark it unavailable for paging.
5 Free the designated area of virtual storage.
FREEMAIN
RMBRANCH
6.1
6 Access next segment.
4
If none
Caller
Output
PVT
PVTFRCF flag
DSPSTI=0
DSPSCT=0
User Segment Table (SGT)
SGTSTE
SGTPAM
System Segment Table (SGT)
SGTSTE
SGTPAM

Diagram 5.35 Destroy Page Table Routine (Module IEAPTCD)

NOTES	ROUTINE NAME	LABEL
1 If the "page-table-in-SQA" option is chosen (bit 0 on in register 1), subpool 245 is used. Otherwise, subpool 255 is used.	Destroy Page Table	IEAPTCD
2 The following parameters are passed to IEAPSIBR: Register 1 - Byte 0 - X'28' (indicates FREE and RELEASE) Bytes 1-3 - starting address (input segment number concatenated with 16 0-bits on the right) Register 2 - Byte 0 - 0 Bytes 1-3 - ending address (input segment number + input segment count) PVTFLAG1 is set with PVTFRCF (X'02') so that FIX counts will be zeroed. (On return, it is set to 0.)		DESTR00
3 If the starting segment number (register 0) = DSPSTI, DSPSTI and DSPSTC (pointed to by PVTSPSTI and PVTSPSTC respectively) are set to 0.		
4 The real address of the PGT is extracted from the STE (CVTSEGB + 4 * segment-number-being-processed) and passed to IEAPTRV in register 1. SGTPAM is set to 1 (invalidated) in both the system and user STEs.		DESTR10
5 The parameters passed to FREEMAIN are: Register 0 - Byte 0 - subpool number Bytes 1-3 - 160 (dec.) Register 1 - Byte 0 - 0 Bytes 1-3 - virtual address of PGT On return, SGTPAM is set to 1 in both STEs.		

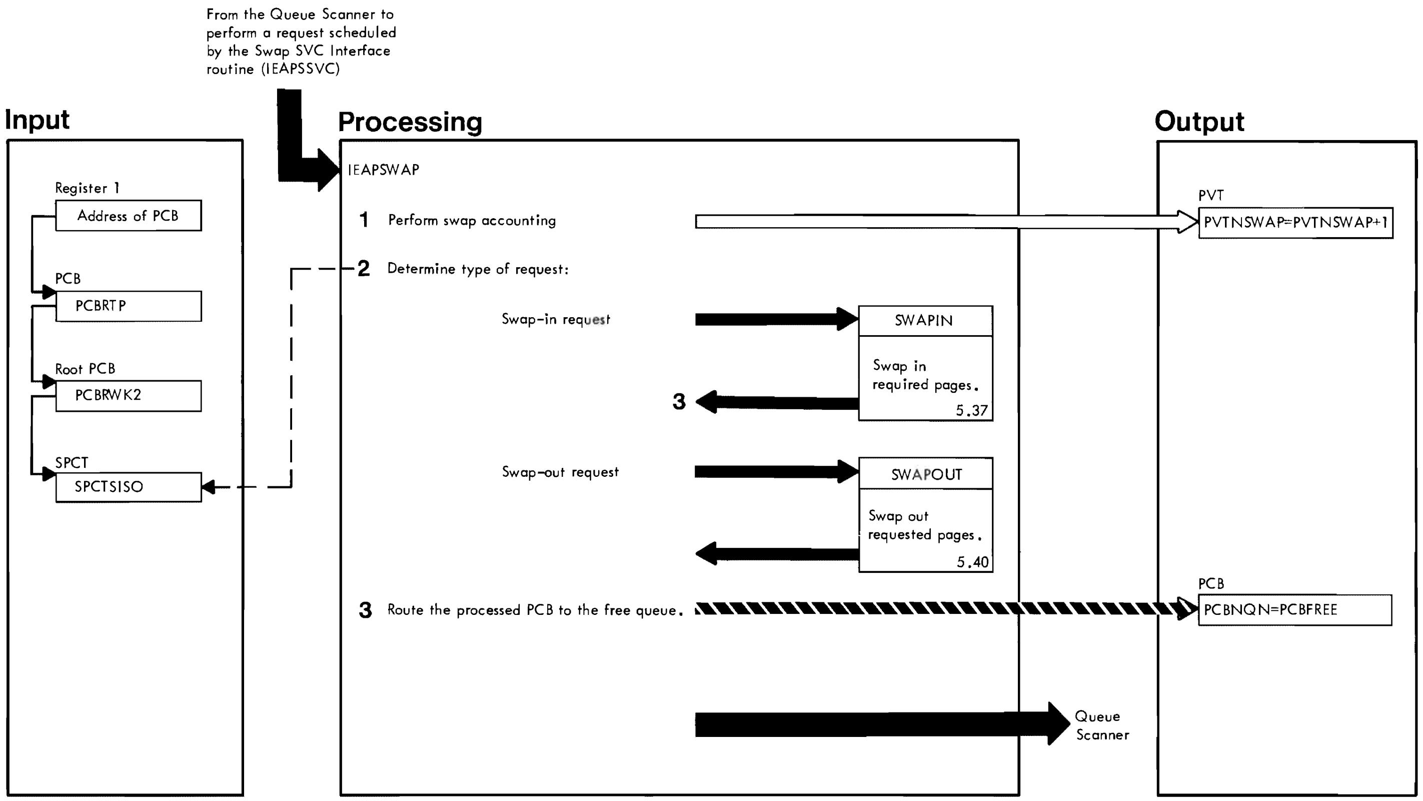
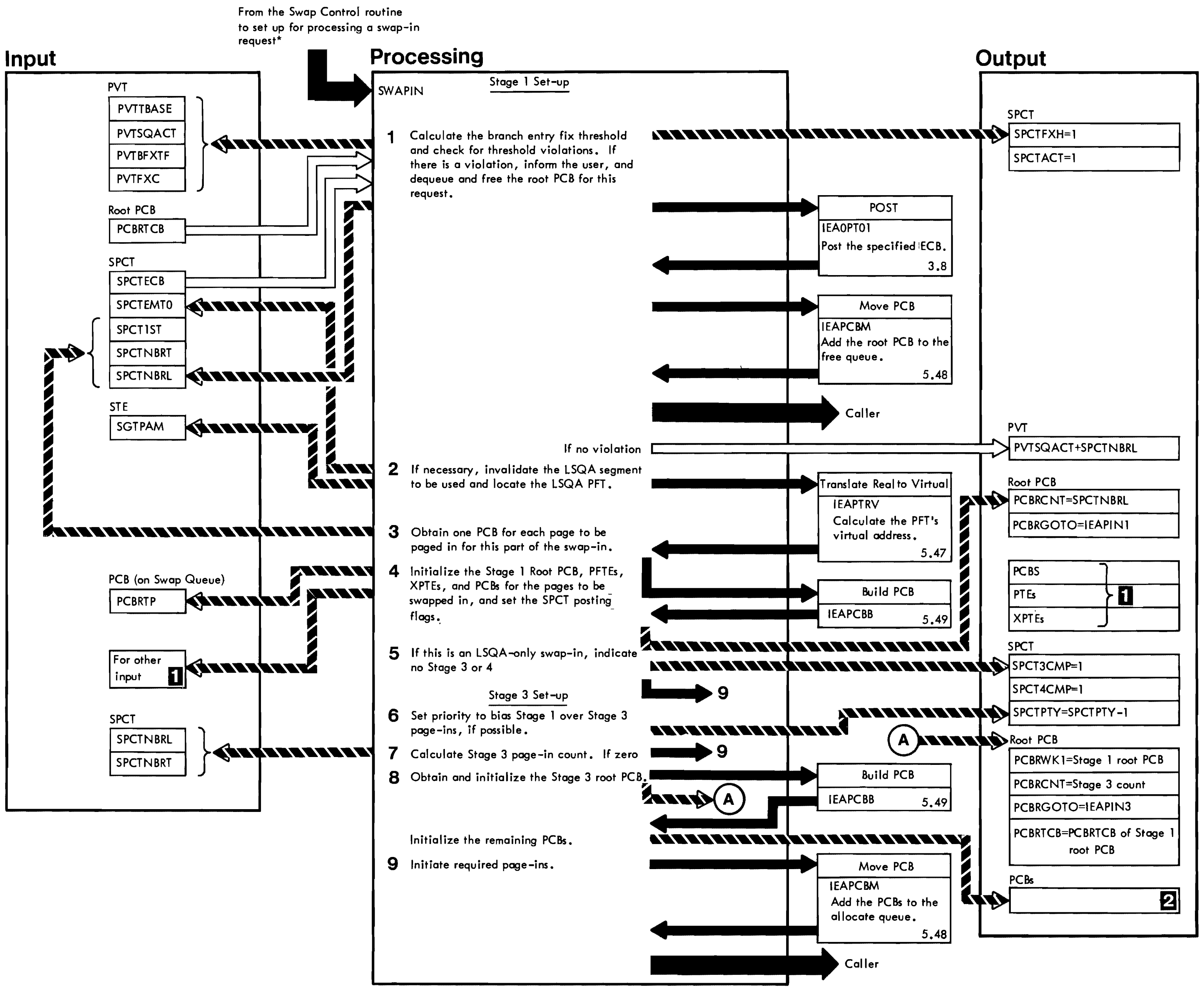

From the Queue Scanner to perform a request scheduled by the Swap SVC Interface routine (IEAPSSVC)
Input
Register 1
Address of PCB
PCB
PCBRTP
Root PCB
PCBRWK2
SPCT
SPCTSISO
Processing
IEAPSWAP
1 Perform swap accounting
2 Determine type of request:
Swap-in request
SWAPIN
Swap in required pages.
5.37
3
Swap-out request
SWAPOUT
Swap out requested pages.
5.40
3 Route the processed PCB to the free queue.
Queue Scanner
Output
PVT
PVTNSWAP=PVTNSWAP+1
PCB
PCBNQN=PCBFREE

Diagram 5.36 Swap Control Routine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
1 Interruptions are disabled before processing begins.	SWAP	IEAPSWAP
2 Interruptions are enabled after Swap-out returns.		

• **Diagram 5.37**
Swap-in Set-up Subroutine


From the Swap Control routine to set up for processing a swap-in request*

Processing

SWAPIN

Stage 1 Set-up

1 Calculate the branch entry fix threshold and check for threshold violations. If there is a violation, inform the user, and dequeue and free the root PCB for this request.

If no violation

2 If necessary, invalidate the LSQA segment to be used and locate the LSQA PFT.

3 Obtain one PCB for each page to be paged in for this part of the swap-in.

4 Initialize the Stage 1 Root PCB, PFTEs, XPTEs, and PCBs for the pages to be swapped in, and set the SPCT posting flags.

5 If this is an LSQA-only swap-in, indicate no Stage 3 or 4

Stage 3 Set-up

6 Set priority to bias Stage 1 over Stage 3 page-ins, if possible.

7 Calculate Stage 3 page-in count. If zero

8 Obtain and initialize the Stage 3 root PCB.

Initialize the remaining PCBs.

9 Initiate required page-ins.

POST
IEAOPT01
Post the specified ECB.
3.8

Move PCB
IEAPCBM
Add the root PCB to the free queue.
5.48

Translate Real to Virtual
IEAPTRV
Calculate the PFT's virtual address.
5.47

Build PCB
IEAPCBB 5.49

Move PCB
IEAPCBM
Add the PCBs to the allocate queue.
5.48

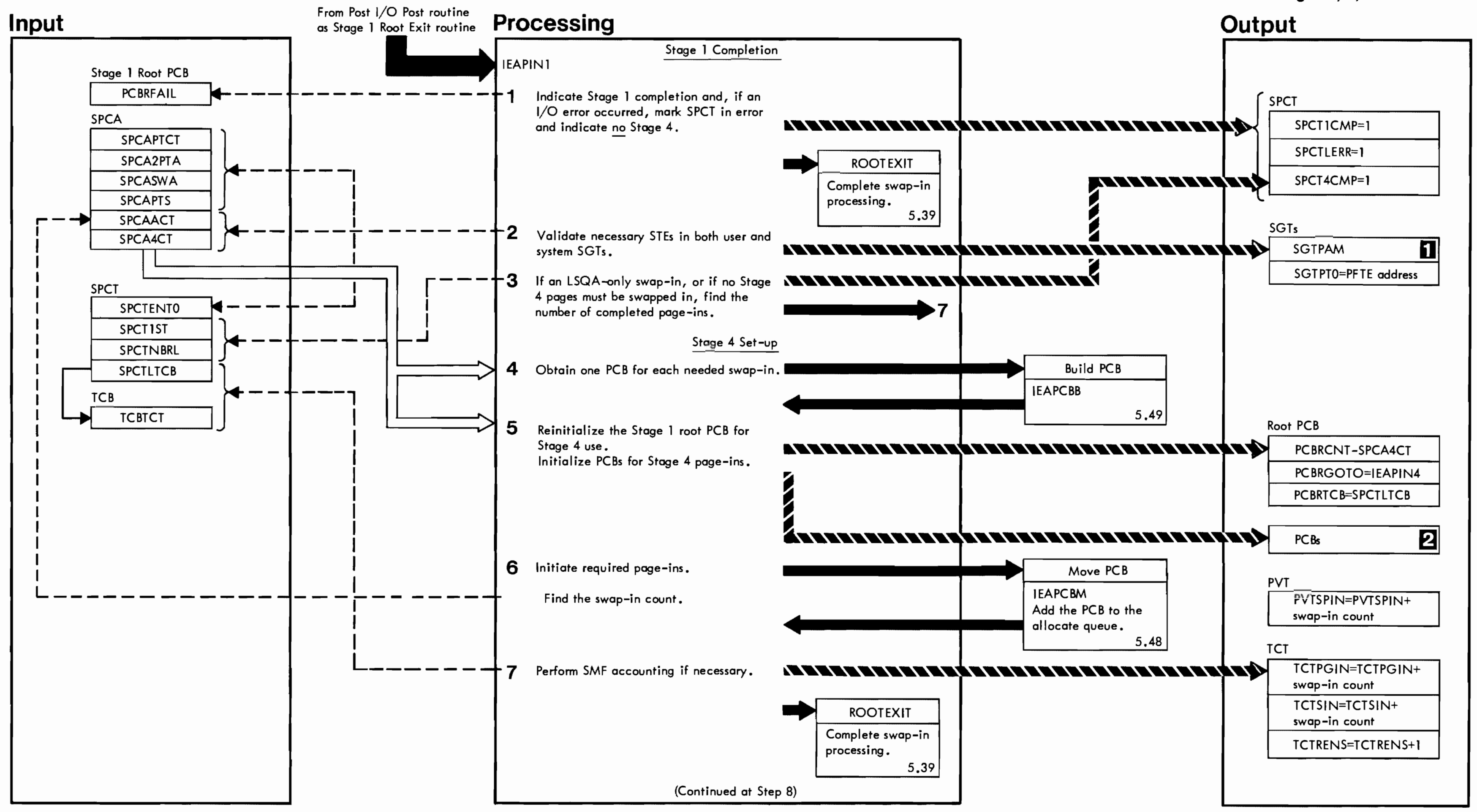
• Diagram 5.37 Swap-in Set-up Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
* Swap-in processing is performed in multiple stages by the following routines: SWAPIN - Stage 1 and 3 set-up. IEAPIN1 - Stage 1 completion and Stage 4 set-up. IEAPIN3 - Stage 3 completion. IEAPIN4 - Stage 4 completion. ROOTEXIT - Swap-in completion. LSQA page-ins are scheduled first (Stage 1) and other page-ins specified by SPCT entries (Stage 3) next. When Stage 1 page-ins are completed, page-ins represented by entries in the SPCA (other than Stage 1 or 3 entries) scheduled (Stage 4). An effective, uninterrupted Swap-in occurs whenever Stage 4 scheduling occurs while Stage 3 I/O is still in progress or when the number of pages to be swapped in is 16 or less (no Stage 4 needed). A 'region ready for restore' indication is given when all SPCT pages have been read in (Stages 1 and 3 complete), but before all Stage 4 page-ins have completed. Thus, Swap-in need not be completed for the whole region before the user can continue processing.	SWAPIN	
1 The branch entry FIX threshold = PVTTBASE-PVTSQACT-PVTBXFTF-SPCTNBRL (the number of LSQA pages required by this swap-in request). If PVTFXC is greater than this threshold, a violation will occur if this request is honored. In this case, the threshold violation flag is set (SPCTFXH=1 and SPCTACT=1) and the user is posted. POST is passed the following parameters: Register 10 - post code = 0. Register 11 - ECB address (from SPCTECB) Register 12 - TCB address (from PCBRTCB). Upon return, the root PCB is disposed of. If the threshold will not be exceeded, the SQA/LSQA count is adjusted to reflect the number of LSQA pages being brought in for this request.		
2 The virtual address of the LSQA PFT is found by using the LSQA segment number (SPCTENT0) as an index to the system STE for the LSQA segment. If the segment is valid (SGTPAM=0), the user either logged off or was abnormally terminated. In this case, LSQA STEs are invalidated and a PTIB instruction is issued (to prevent the hardware from giving unwanted information). The real address of the PFT is extracted from the STE and IEAPTRV is called to return its virtual equivalent.		
3 If this is an LSQA-only swap-in (SPCT1ST=1), SPCTNBRL indicates the number of PCBs needed. If SPCT1ST=0, SPCTNBRT is used.		

NOTES	ROUTINE NAME	LABEL
4 The Root PCB is pointed to by PCBRTP in the PCB on the swap queue for this request.		
5 If this is an LSQA-only swap request, Stages 3 and 4 are marked complete (SPCT3CMP=1 and SPCT4CMP=1).		
6 The priority of Stage 1 pages is reduced by 1, if possible.		
7 The Stage 3 page-in count = SPCTNBRT-SPCTNBRL.		
9 When the Stages complete, Page I/O Post will give control to the Root Exit routine pointed to by the PCBRGOTO field of the completing root (IEAPIN1 for the Stage 1 root and IEAPIN3 for the Stage 3 root).		
1 PCBs and table entries are set up as follows: • First, the LSQA PFTEs are zeroed except that PGTPVM=1 in each. • The XPT for this PGT is zeroed. • The page number (from SPCTVM) is multiplied by 2 and added to the PGT address to find the PTE. The XPTE address = the PTE address + 32 + 8*SPCTVM. • PGTPAM is set to 1 to indicate page assigned. • SPCTXPT (the XPT address) is moved to the XPTE. • XPTXAV is set to 1 to indicate page assigned. • If this is a warm-start page (SPCTFWST=1), XPTLPA is set to 1 to indicate that the page cannot be destroyed. • The PCB is set as follows: • PCBNQN=PCBALLOC to route the PCB to the real storage allocation queue. • PCBIOI=1 to indicate page-in. • PCBRTP=the Root PCB address. • PCBFXC=1 to prevent the page from being paged out (by replacement) when the page is brought in. • PCBPTY=SPCTPTY. • PCBXPT=XPTE address calculated above. • PCBPTE=PTE address calculated above. • PCBVBN=SPCTVM. • The next PCB is accessed via PCBFQP. The next SPCT entry is found and this loop repeats for each PCB and entry. • When the last entry is completed, the swap post flags (SPCTFL2, SPCTLERR, SPCTERR, SPCTFXH, SPCTOERR, SPCTTHER) are zeroed.		

NOTES	ROUTINE NAME	LABEL
2 The remaining PCBs are set up from the remaining (that is, non-LSQA) SPCT entries as follows: • PCBNQN=PCBALLOC • PCBIOI=1 • PCBRTP=Stage 3 root address • PCBFXC=1 • PCBPTY=adjusted SPCTPTY • PCBXPT=address of a dummy 8-byte constant with only XPTXAV=1 set • PCBDADDF=1 • PCBPTE=address of SPCTPTE of current SPCT entry • PCBVBN=SPCTVM • PCBDADD=SPCTXPT SPCTPTE is set up to appear as an invalid, assigned PTE.		

Diagram 5.38 (Steps 1-7) Swap-in Completion Routines for Stages 1, 3, and 4



Stage 1 Completion

1 Indicate Stage 1 completion and, if an I/O error occurred, mark SPCT in error and indicate no Stage 4.

2 Validate necessary STEs in both user and system SGTs.

3 If an LSQA-only swap-in, or if no Stage 4 pages must be swapped in, find the number of completed page-ins.

Stage 4 Set-up

4 Obtain one PCB for each needed swap-in.

5 Reinitialize the Stage 1 root PCB for Stage 4 use. Initialize PCBs for Stage 4 page-ins.

6 Initiate required page-ins.

Find the swap-in count.

7 Perform SMF accounting if necessary.

(Continued at Step 8)

Diagram 5.38 (Steps 1-7) Swap-in Completion Routines for Stages 1, 3, and 4
(Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
1 If PCBRFAIL=1, an I/O error occurred.	IEAPIN1	
2 The LSQA STEs in both the system and user SGTs are located by using the segment number in byte 0 of SPCTENT0 as an index into each SGT. SPCA entries are looped through as follows: • SPCAPTS*4 is used as an index into the SGTs. • The virtual address of the PFT is translated to a real address (via an LRA instruction) and stored in SGTPTO.		
3 For an LSQA-only request (SPCT1ST=1), SPCTNBRL is used for SMF accounting. If no Stage 4 pages are to be swapped in (SPCT4CT=0), Stage 4 is marked complete and the active swap-in count (SPCAACT) is used for SMF accounting.		
4 SPCT4CT contains the number of PCBs needed.		
5 The root address is passed by Page I/O Post.		
7 If TCBTCT=0, no SMF accounting is performed.		
$\boxed{1}$ • The system STE is validated (SGTPAM=0). • If this is the SWA segment, the user STE is validated. Otherwise, it is invalidated.		
$\boxed{2}$ PCBs and table entries are set up as follows: • SPCANTPT is used to index the appropriate SPCAPTA. • The virtual address of the PGT for this entry (SPCAPGTA) is picked up. SPCAPGTA*2+SPCANTVM = the PTE address for this SPCA entry. • The XPTE address = the PTE address + SPCANTXP. • PCBPTE is set to the calculated PTE address and PCBXPT to the calculated XPTE address. • PCBNQN is set to PCBALLOC. • PCBVBN is set to SPCANTVM. • PCBIOI is set to 1. • PCBPTY is set to the adjusted priority (SPCTPTY). (SPCTPTY is adjusted by 1 to bias Stage 3 over Stage 4 page-ins.		

Diagram 5.38 (Steps 8-10) Swap-in Completion Routines for Stages 1, 3, and 4

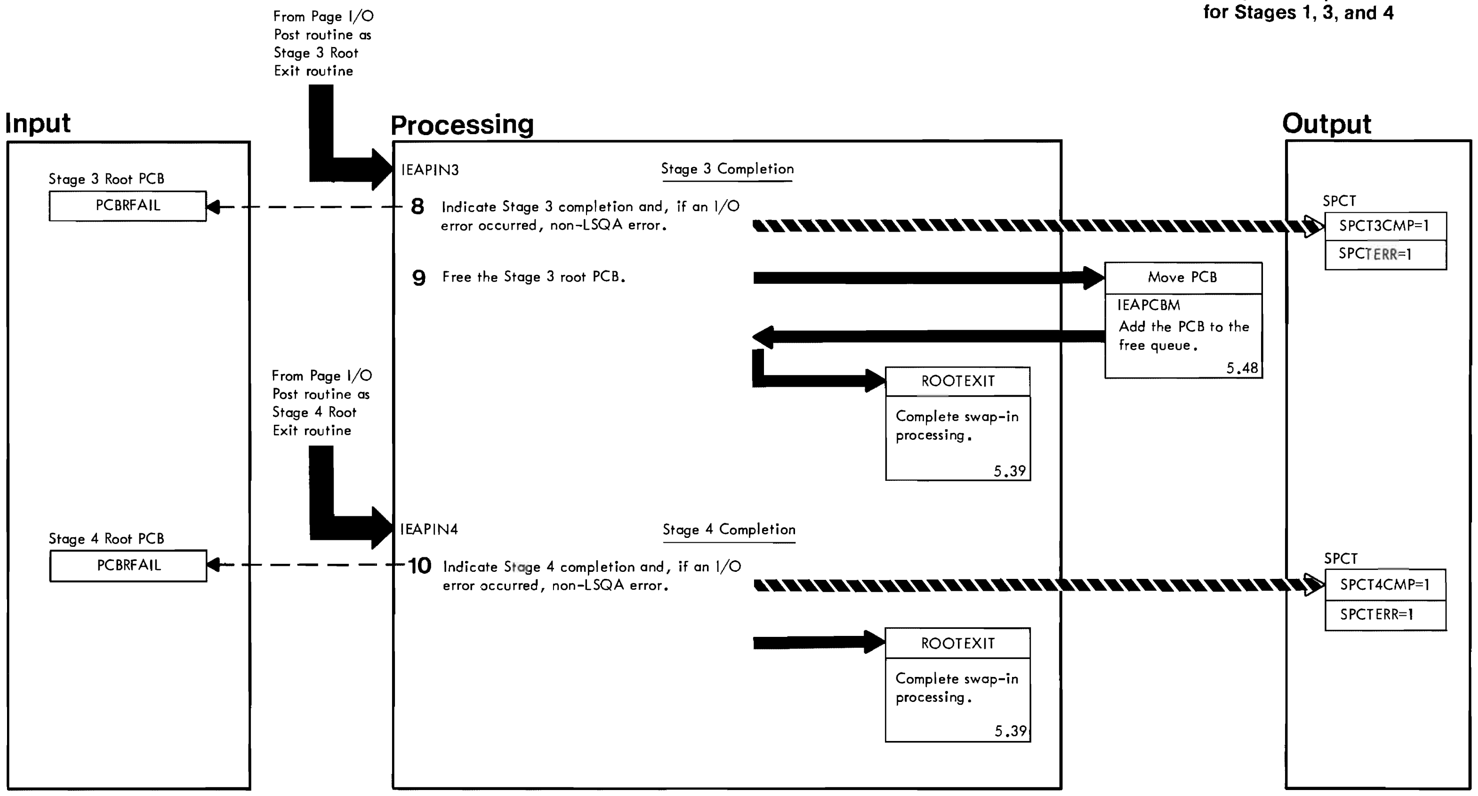
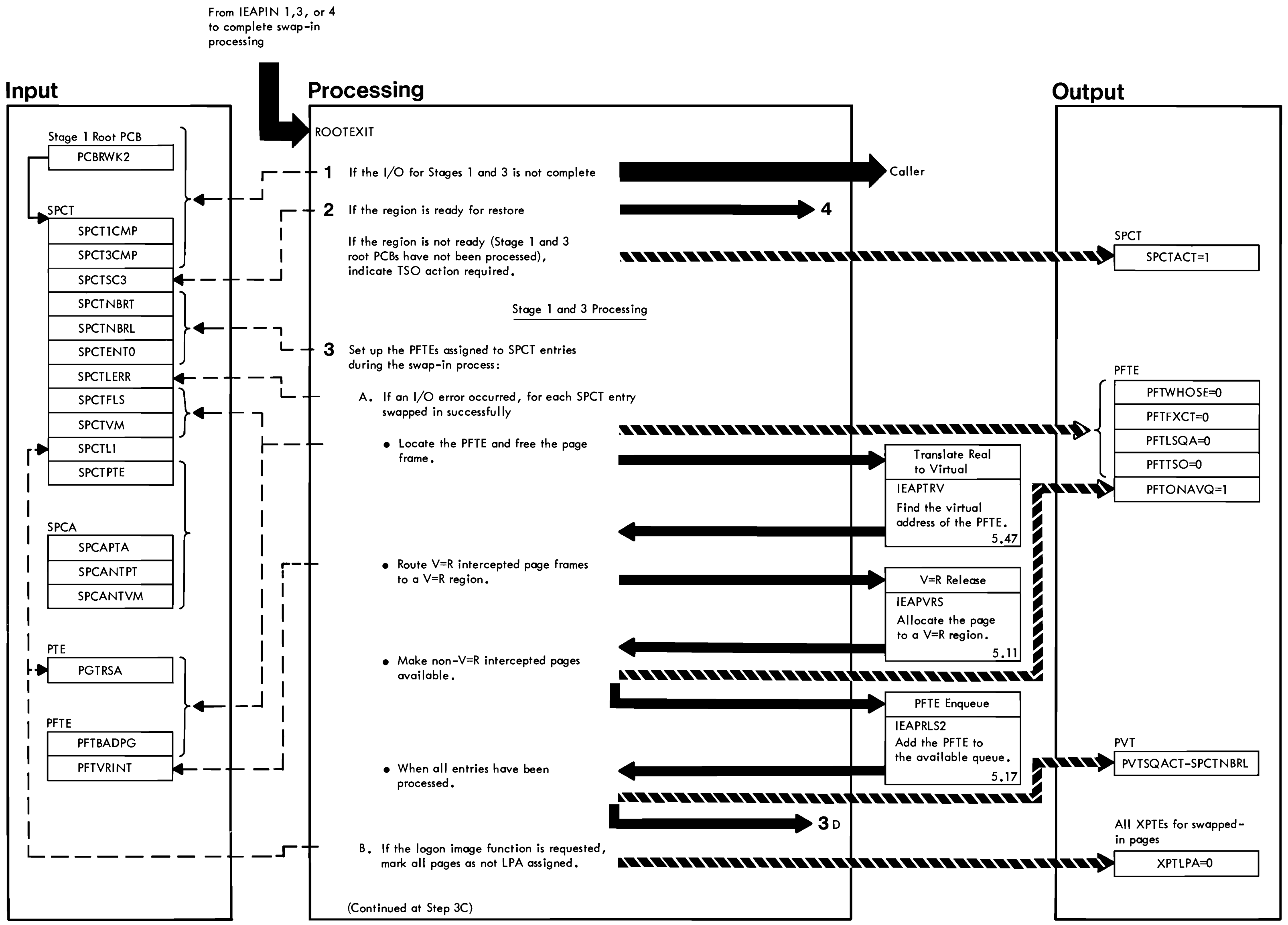

From Page I/O Post routine as Stage 3 Root Exit routine
Input
Processing
Output
Stage 3 Root PCB
PCBRFAIL
IEAPIN3
Stage 3 Completion
8 Indicate Stage 3 completion and, if an I/O error occurred, non-LSQA error.
9 Free the Stage 3 root PCB.
SPCT
SPCT3CMP=1
SPCTERR=1
Move PCB
IEAPCBM
Add the PCB to the free queue.
5.48
ROOTEXIT
Complete swap-in processing.
5.39
From Page I/O Post routine as Stage 4 Root Exit routine
Stage 4 Root PCB
PCBRFAIL
IEAPIN4
Stage 4 Completion
10 Indicate Stage 4 completion and, if an I/O error occurred, non-LSQA error.
SPCT
SPCT4CMP=1
SPCTERR=1
ROOTEXIT
Complete swap-in processing.
5.39

Diagram 5.38 (Steps 8-10) Swap-in Completion Routines for Stages 1, 3, and 4 (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
8 Registers are saved for ROOTEXIT. If PCBRFAIL=1, an I/O error occurred for a Stage 3 page-in.	IEAPIN3	
10 Registers are saved for ROOTEXIT. If PCBRFAIL=1, an I/O error was encountered during Stage 4 page-in processing.	IEAPIN4	

• **Diagram 5.39 (Steps 1-3B) Swap-in Completion Subroutine**



1 If the I/O for Stages 1 and 3 is not complete

2 If the region is ready for restore

If the region is not ready (Stage 1 and 3 root PCBs have not been processed), indicate TSO action required.

Stage 1 and 3 Processing

3 Set up the PFTEs assigned to SPCT entries during the swap-in process:

A. If an I/O error occurred, for each SPCT entry swapped in successfully

- Locate the PFTE and free the page frame.
- Route V=R intercepted page frames to a V=R region.
- Make non-V=R intercepted pages available.
- When all entries have been processed.

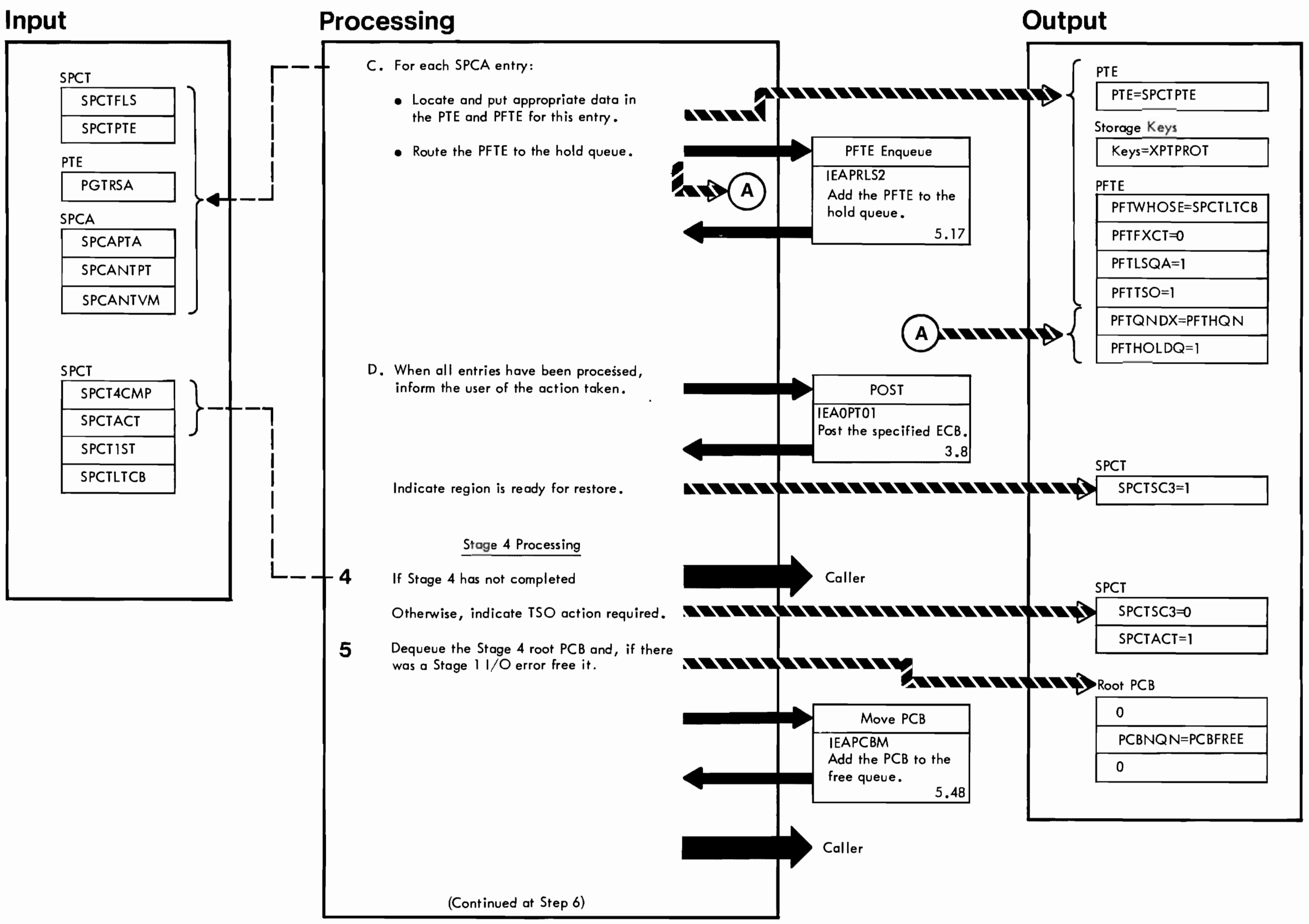
B. If the logon image function is requested, mark all pages as not LPA assigned.

(Continued at Step 3C)

Diagram 5.39 (Steps 1-3B) Swap-in Completion Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
1 PCBRWK2 of the Stage 1 root points to the SPCT for this request.	ROOTEXIT	
2 If SPCTSC3=1, the region is ready for Restore.		
3 For an LSQA-only request, SPCTNBRL is used as a loop counter. Otherwise, SPCTNBRT is used.		
3A If SPCTLERR=1, an I/O error occurred.		
• If SPCTFLS=1, this is an LSQA entry. The PFTE is found by using SPCTVM. If this is not an LSQA entry, the PFTE is found by using PGTRSA of the dummy PTE.		
• PFTFXCT, PFTLSQA, PFTTSO, and PFTWHOSE are zeroed.		
• If the page has been marked not usable by RMS (PFTBADPG=1), nothing more is done to the page.		
• If the page is marked for V=R interception (PFTVRINT=1), it is given to V=R Release to be allocated to a pending V=R region request.		
• Otherwise, the page is made available via PFTE Enqueue and PFTONAVQ is set to 1.		
• When the last entry has been processed, PVTSQACT is decremented by SPCTNBRL and SPCT1ST is set to 1.		
3B If SPCTLI=1, XPTLPA is reset for all pages swapped in as part of this request by indexing through the 16 PTEs per segment as defined in all SPCAPTAs for the region. This allows the external back-up group of Logon Image pages to be freed at "STOP TSO" time.		

• Diagram 5.39 (Steps 3C-5)
Swap-in Completion Subroutine



C. For each SPCA entry:

- Locate and put appropriate data in the PTE and PFTE for this entry.
- Route the PFTE to the hold queue.

D. When all entries have been processed, inform the user of the action taken.

Indicate region is ready for restore.

<u>Stage 4 Processing</u>

4 If Stage 4 has not completed

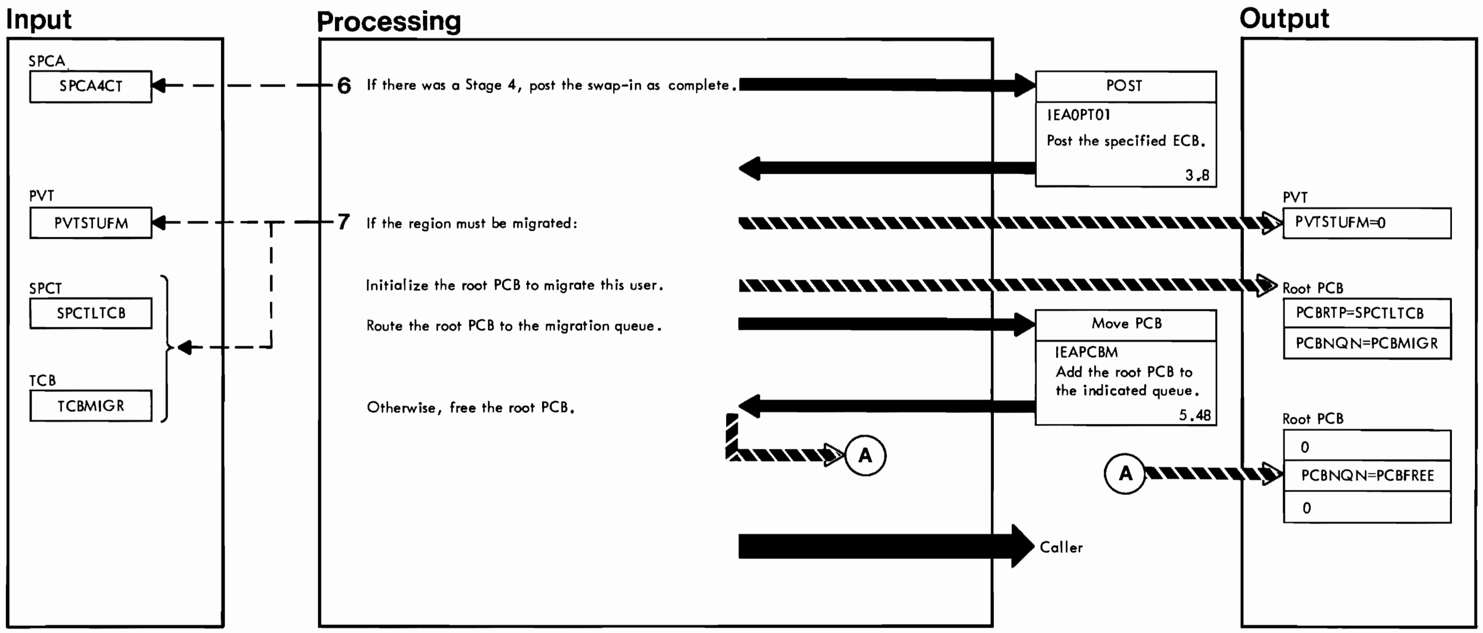
Otherwise, indicate TSO action required.

5 Dequeue the Stage 4 root PCB and, if there was a Stage 1 I/O error free it.

(Continued at Step 6)

Diagram 5.39 (Steps 3C-5) Swap-in Completion Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
3C • The PTE address for the current entry is found by indexing to SPCAPTA via SPCANTPT and adding the page number (from SPCANTVM) times 2. • If the page is not for LSQA (SPCTFLS=0), SPCTPTE (the dummy PTE) is moved into the real PTE. • PGTRSA is used to calculate the PFTE address for the page. PFTFXCT is then set to 0 and PFTWHOSE is set to SPCTLTCB. • If the page is for LSQA, PFTLSQA and PFTTSO are set to 1. If the page is not for LSQA, the storage keys are set from XPTPROT. 3D The following parameters are passed to POST: Register 10 - post code = 0. Register 11 - ECB address (from SPCTECB). Register 12 - TCB address (from PCBRTCB). Upon return from POST, SPCTSC3 is set to 1 to indicate that the Stage 1 and 3 roots have been processed.	ROOTEXIT	
4 If SPCT4CMP=1, the swap-in is marked complete (SPCTSC4 =1). If SPCTACT=0, SPCTSC3 is zeroed and SPCTACT is set to 1.		ROOTST4



6 If there was a Stage 4, post the swap-in as complete.

7 If the region must be migrated:

Initialize the root PCB to migrate this user.

Route the root PCB to the migration queue.

Otherwise, free the root PCB.

Diagram 5.39 (Steps 6-7) Swap-in Completion Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
6 If the Stage 4 count (SPCA4CT) is not 0, a second posting to indicate swap-in complete is required. Parameters are as above.	ROOTEXIT	
7 The region must be migrated if the migrate select flag is on (PVTSTUFM=1) and the region is not migrated (TCBMIGR=0).		

• **Diagram 5.40 (Steps 1-6) Swap-out Control Subroutine**

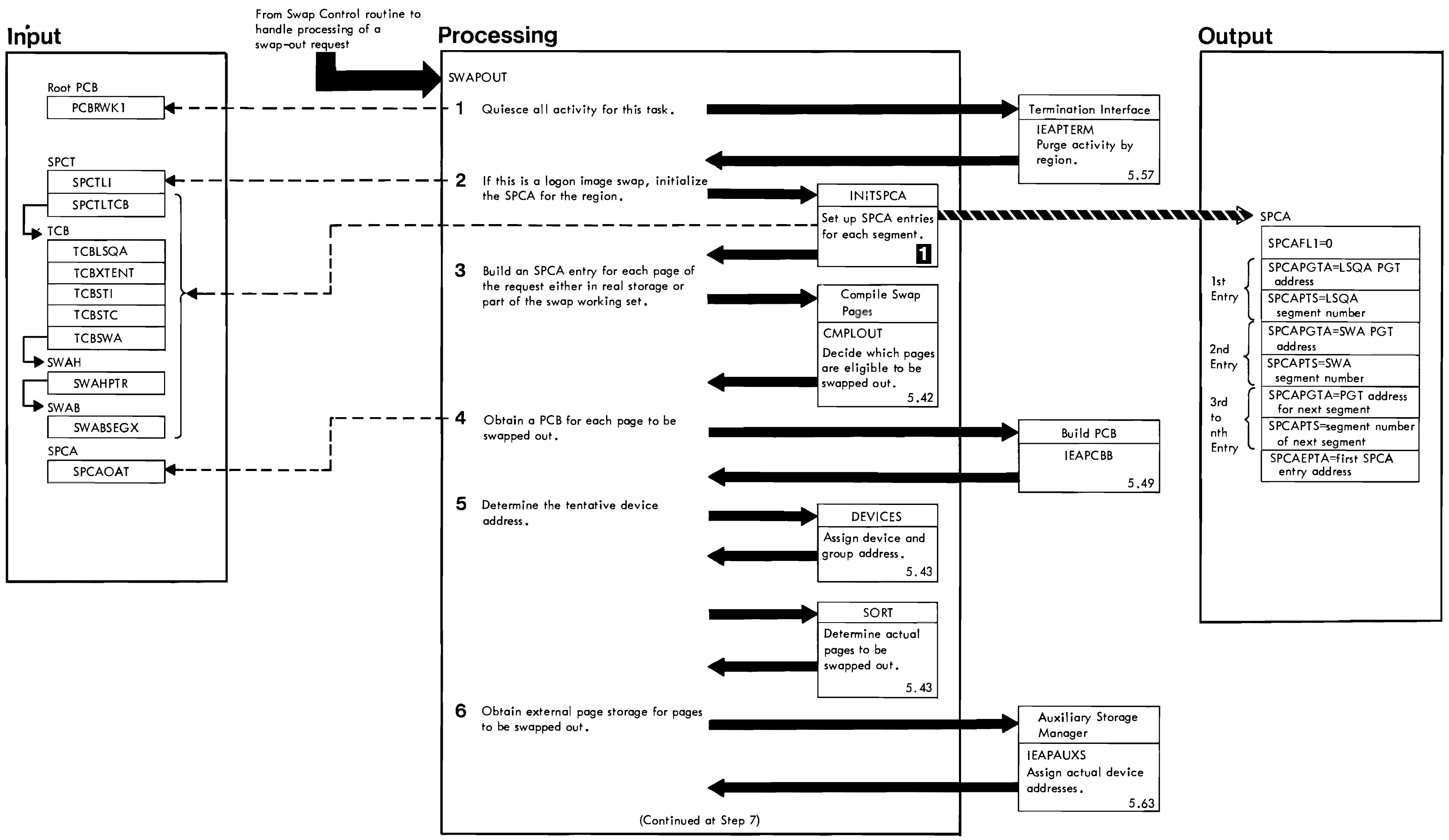
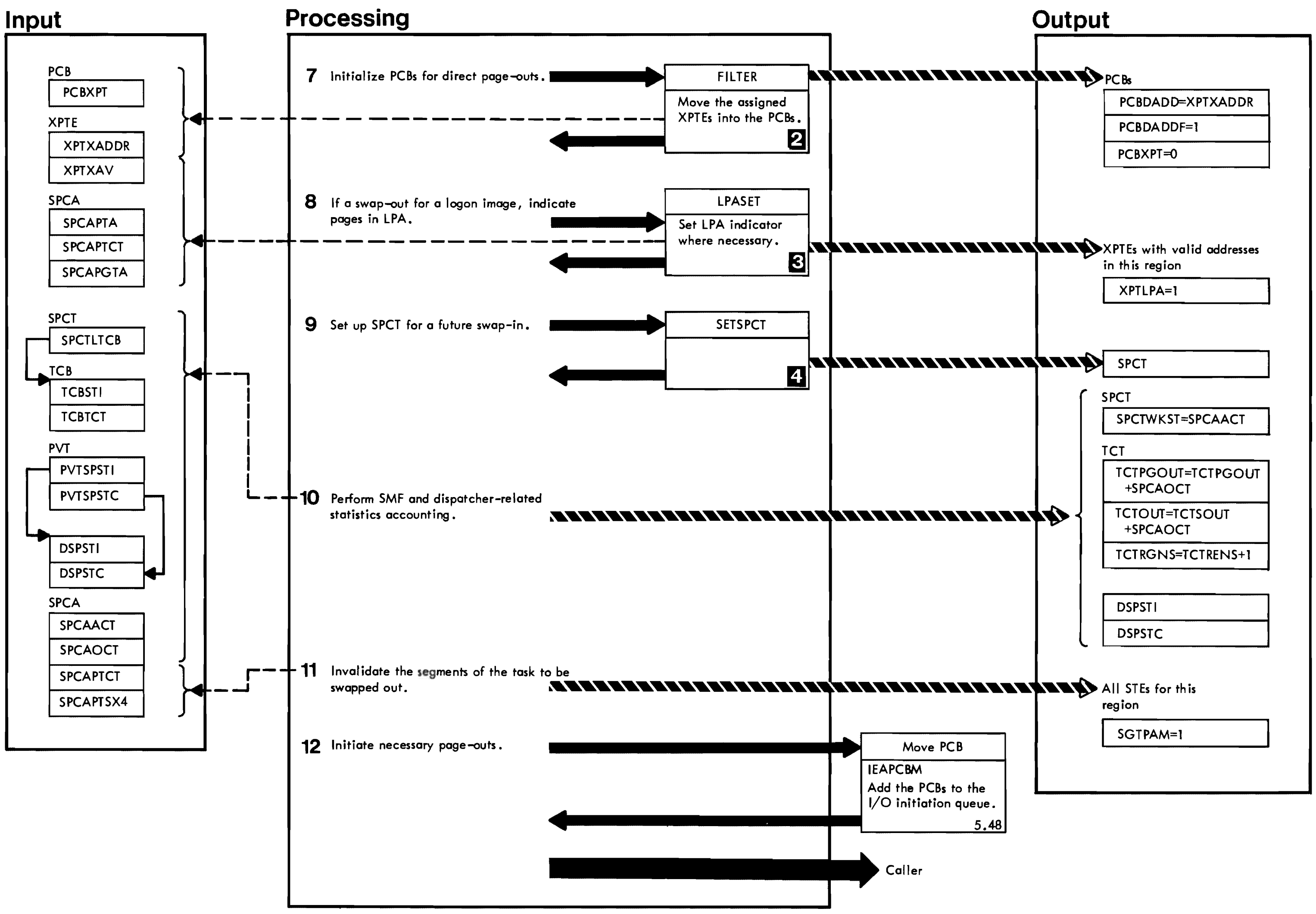

Input
Root PCB
PCBRWK1
SPCT
SPCTLI
SPCTLTCB
TCB
TCBLSQA
TCBXTENT
TCBSTI
TCBSTC
TCBSWA
SWAH
SWAHPTR
SWAB
SWABSEGX
SPCA
SPCAOAT
From Swap Control routine to handle processing of a swap-out request
Processing
SWAPOUT
1 Quiesce all activity for this task.
2 If this is a logon image swap, initialize the SPCA for the region.
3 Build an SPCA entry for each page of the request either in real storage or part of the swap working set.
4 Obtain a PCB for each page to be swapped out.
5 Determine the tentative device address.
6 Obtain external page storage for pages to be swapped out.
(Continued at Step 7)
Termination Interface
IEAPTERM
Purge activity by region.
5.57
INITSPCA
Set up SPCA entries for each segment.
1
Compile Swap Pages
CMPLOUT
Decide which pages are eligible to be swapped out.
5.42
Build PCB
IEAPCBB
5.49
DEVICES
Assign device and group address.
5.43
SORT
Determine actual pages to be swapped out.
5.43
Auxiliary Storage Manager
IEAPAUXS
Assign actual device addresses.
5.63
Output
SPCA
SPCAFL1=0
1st Entry
SPCAPGTA=LSQA PGT address
SPCAPTS=LSQA segment number
2nd Entry
SPCAPGTA=SWA PGT address
SPCAPTS=SWA segment number
3rd to nth Entry
SPCAPGTA=PGT address for next segment
SPCAPTS=segment number of next segment
SPCAEPTA=first SPCA entry address

Diagram 5.40 (Steps 1-6) Swap-out Control Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
1 The parameters passed to IEAPTERM are: Register 0 - TCB address (from PCBRWK1). Register 1 - 8.	SWAPOUT	
4 The swap-out count (SPCAOAT) indicates the number or PCBs needed.		
6 A PTLB instruction is issued before the call to the Auxiliary Storage Manager. Registers are set up to make it appear as if the call is from the Queue Scanner as follows: Register 1 - address of the first PCB. Register 3 - PVT address.		
1 INITSPCA initializes the SPCA PGT address entries by calculating segment numbers and PGT addresses for each segment in the region to be swapped out. Find Page (IEAPFP2) is called to give the virtual addresses of the PGTs. If SWAHPTR=0, there is no SWA region and the special second entry (see output above) is not set up.	INITSPCA	



Input

PCB
- PCBXPT

XPTE
- XPTXADDR
- XPTXAV

SPCA
- SPCAPTA
- SPCAPTCT
- SPCAPGTA

SPCT
- SPCTLTCB

TCB
- TCBSTI
- TCBTCT

PVT
- PVTSPSTI
- PVTSPSTC

- DSPSTI
- DSPSTC

SPCA
- SPCAACT
- SPCAOCT
- SPCAPTCT
- SPCAPTSX4

Processing

7 Initialize PCBs for direct page-outs.

FILTER
Move the assigned XPTEs into the PCBs. 2

8 If a swap-out for a logon image, indicate pages in LPA.

LPASET
Set LPA indicator where necessary. 3

9 Set up SPCT for a future swap-in.

SETSPCT
4

10 Perform SMF and dispatcher-related statistics accounting.

11 Invalidate the segments of the task to be swapped out.

12 Initiate necessary page-outs.

Move PCB
IEAPCBM
Add the PCBs to the I/O initiation queue.
5.48

Caller

Output

PCBs
- PCBDADD=XPTXADDR
- PCBDADDF=1
- PCBXPT=0

XPTEs with valid addresses in this region
- XPTLPA=1

- SPCT

SPCT
- SPCTWKST=SPCAACT

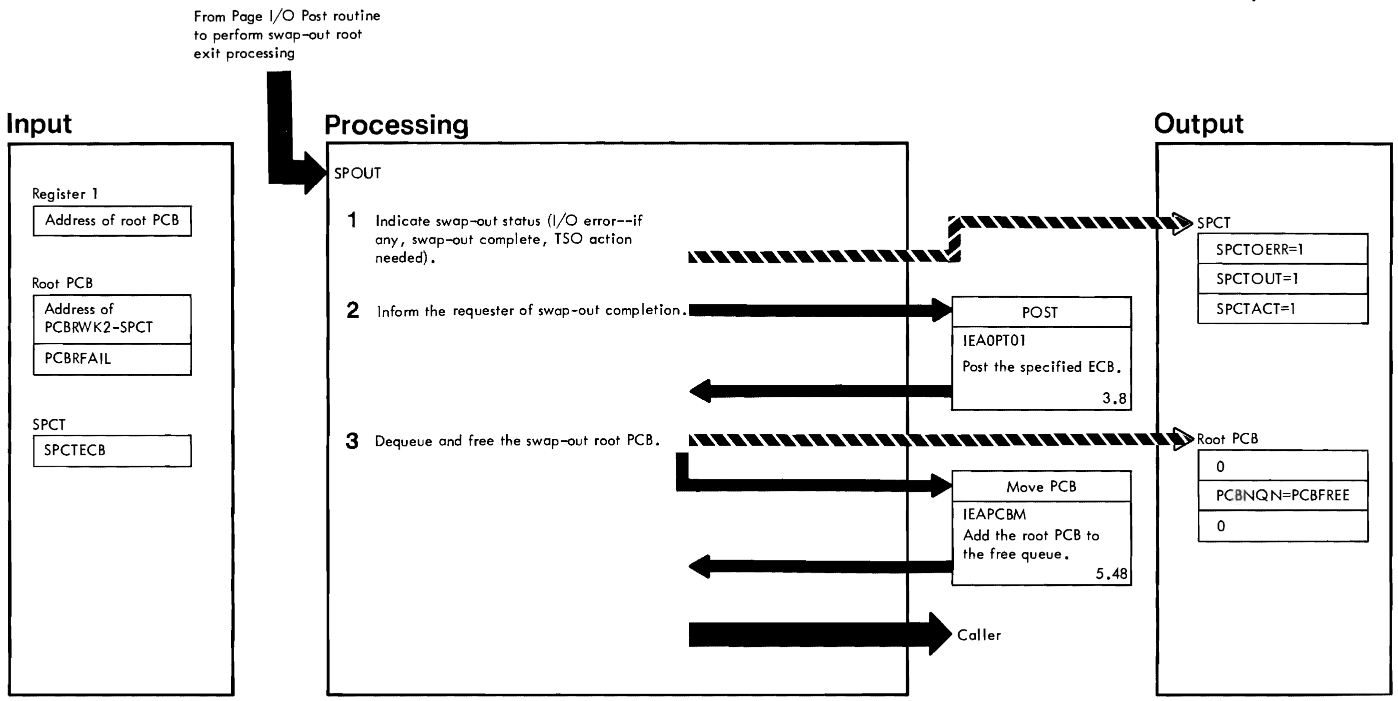
TCT
- TCTPGOUT=TCTPGOUT +SPCAOCT
- TCTOUT=TCTSOUT +SPCAOCT
- TCTRGNS=TCTRENS+1

- DSPSTI
- DSPSTC

All STEs for this region
- SGTPAM=1

Diagram 5.40 (Steps 7-12) Swap-out Control Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
7	SWAPOUT	
10 If TCBSTI=DSPSTI, DSPSTI and DSPSTC are zeroed. The swap-out count (PVTSPOUT) is incremented by the number of pages to be swapped out for this request (SPCAOCT). If TCBTCT=0, no SMF accounting is performed.		
11 SPACPTCT indicates how many STEs must be invalidated. SPCAPTS*4 is used as the index into the system and user SGTs. Entries are processed in reverse order so that LSQA entries are invalidated last.		
12 A PTLB instruction is issued before the call to Move PCB. PCBNQN has been set to PCBINIT by the Auxiliary Storage Manager.		
$\boxed{2}$ FILTER moves the XPTEs actually assigned by the Auxiliary Storage Manager into the PCBs for the swap-out. The XPTE represented by each PCB is picked up from PCBXPT. The external page address (XPTXADDR) is moved to PCBDADD, PCBDADDF is set to 1, PCBXPT is zeroed.		
$\boxed{3}$ LPASET sets the "do-not-destroy" flag (XPTLPA) in every XPTE with a valid address in the region to be swapped out. The XPTE is found by adding 32 to the address of the PTE (located through SPCAPGTA). If XPTXAV=1 in the entry, XPTLPA is set to 1.	LPASET	
$\boxed{4}$ SETSPCT loops through all SPCA entries setting necessary flags and counts in the SPCT to be used when the region is swapped in. 1 SPCAAPT is loaded into an entry count register. 2 The LSQA SPCT and SPCT entry counts are zeroed (SPCTNBRL and SPCTNBRT respectively). For each SPCT entry: 3 SPCTVM is set to SPCANTVM. 4 SPCANTSP is set to 1. 5 The XPTE address is calculated ((SPCAPGTA+2)*SPCANTVM +SPCANTXP). XPTXADDR is moved to SPCTXPT. 6 If the PTE number (SPCANTPT) = 0: • SPCTFLS is set to 1 to indicate an LSQA page. • SPCTNBRL is incremented by 1. • SPCTFWST is set to 1 (if XPTLPA=1). 7 SPCTNBRT is incremented by 1 and the "entry count register" (from above) is decremented by 1. 8 If the "entry count register" ≠ 0 or SPCTNBRT ≠ 16, repeat from "3" for the next SPCA and SPCT entries. 9 When all entries have been handled, PVTSQACT is decremented by SPCTNBRL. 10 The residual SPCA count (all pages above 16) is stored in SPCA4CT (Stage 4 page-in count) and SPCAF4A is set to point to the first Stage 4 entry.	SETSPCT	



1 Indicate swap-out status (I/O error--if any, swap-out complete, TSO action needed).

2 Inform the requester of swap-out completion.

3 Dequeue and free the swap-out root PCB.

Diagram 5.41 Swap-out Completion Routine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
1 If PCBRFAIL=1, SPCTOERR is set to 1.	SROUT	
2 The ECB address from SPCTECB is passed to POST.		

Diagram 5.42 (Steps 1-5) Swap-out—CMPLOUT Subroutine (Deciding Which Pages Are to Be Swapped Out)

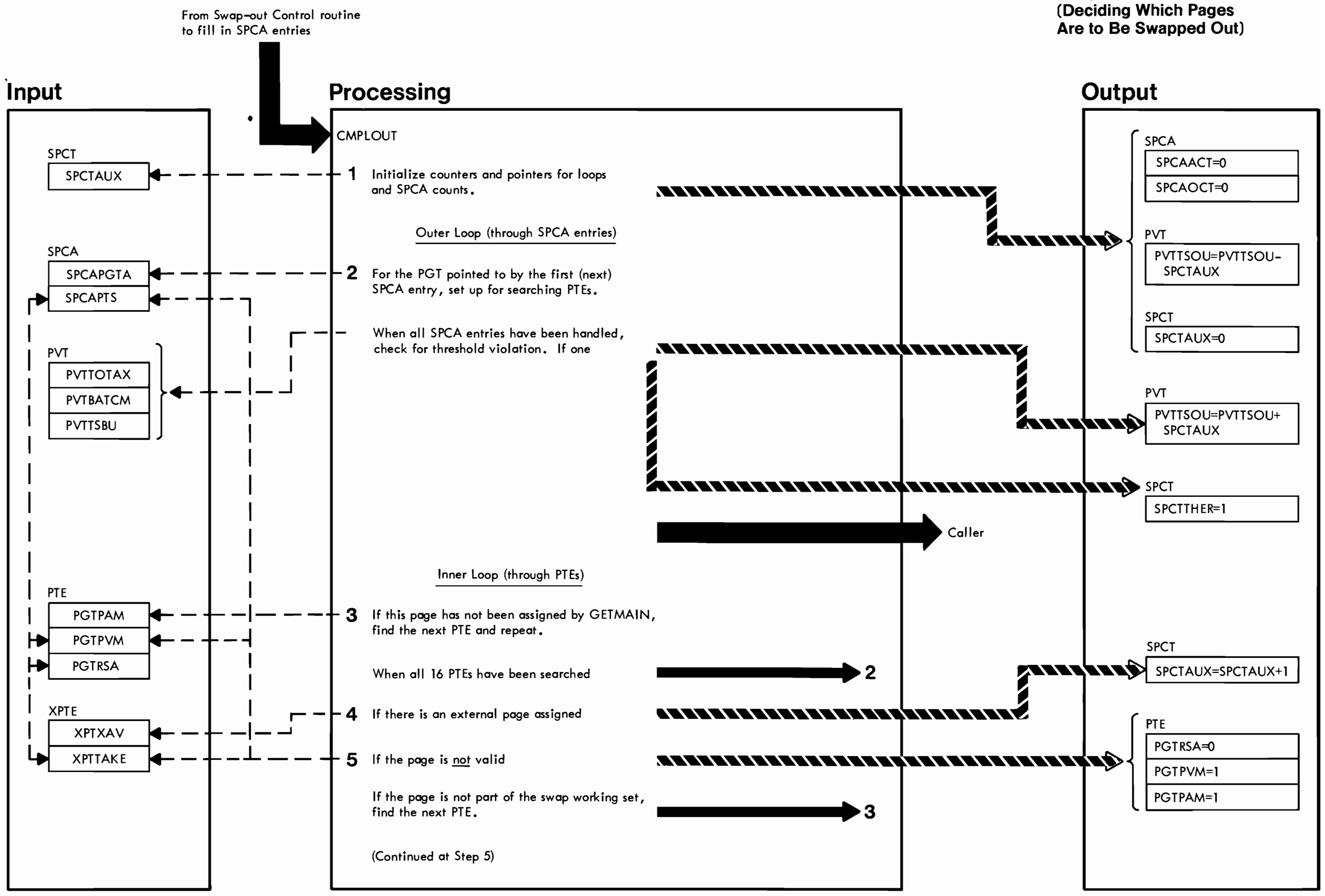
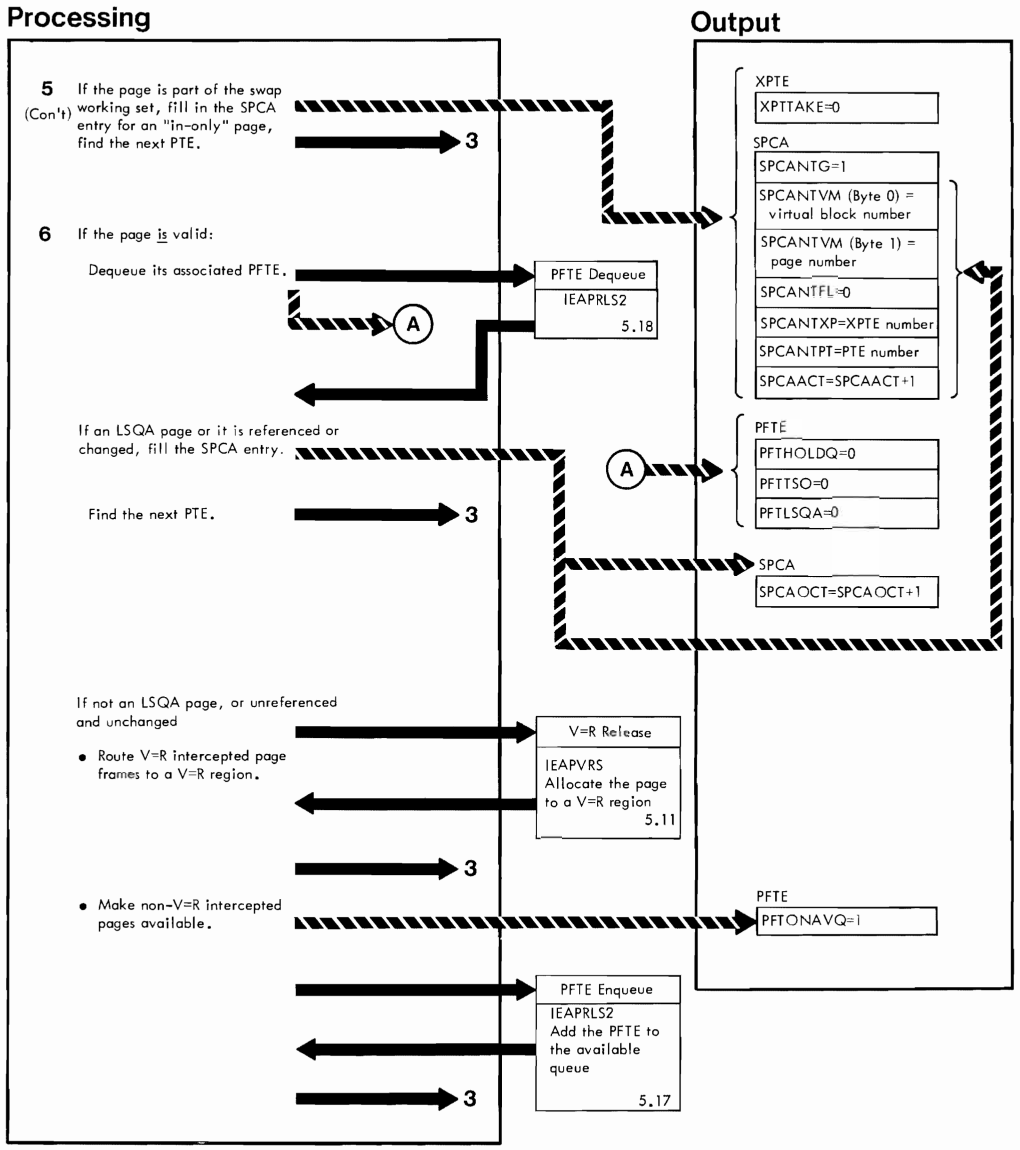

From Swap-out Control routine to fill in SPCA entries
Input
SPCT
SPCTAUX
SPCA
SPCAPGTA
SPCAPTS
PVT
PVTTOTAX
PVTBATCM
PVTTSBU
PTE
PGTPAM
PGTPVM
PGTRSA
XPTE
XPTXAV
XPTTAKE
Processing
CMPLOUT
1 Initialize counters and pointers for loops and SPCA counts.
Outer Loop (through SPCA entries)
2 For the PGT pointed to by the first (next) SPCA entry, set up for searching PTEs.
When all SPCA entries have been handled, check for threshold violation. If one
Caller
Inner Loop (through PTEs)
3 If this page has not been assigned by GETMAIN, find the next PTE and repeat.
When all 16 PTEs have been searched
2
4 If there is an external page assigned
5 If the page is not valid
If the page is not part of the swap working set, find the next PTE.
3
(Continued at Step 5)
Output
SPCA
SPCAACT=0
SPCAOCT=0
PVT
PVTTSOU=PVTTSOU–SPCTAUX
SPCT
SPCTAUX=0
PVT
PVTTSOU=PVTTSOU+SPCTAUX
SPCT
SPCTTHER=1
SPCT
SPCTAUX=SPCTAUX+1
PTE
PGTRSA=0
PGTPVM=1
PGTPAM=1

Diagram 5.42 (Steps 1-5) Swap-out - CMPLOUT Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
1 SPCA counts are zeroed and the count of external pages used by TSO (PVTTSOU) is decremented by the number previously used by this region.	CMPLOUT	
2 A PTE pointer is initialized from SPCAPGTA and 32 is added to it for the XPTE pointer.		
The new SPCT external storage count (SPCTAUX) is added to PVTTSOU to determine how may external pages TSO is now using. The amount available to TSO is calculated (PVTTOTAX-PVTBATCM-PVTTSBU). If PVTTSOU is greater than the result, the threshold has been violated.		
4 If an external page has been assigned (XPTXAV=1), SPCTAUX is incremented.		
5 If PGTPVM=1, the page is invalid. If XPTTAKE≠1, no further action is taken. (XPTTAKE is set by IEAPTERM.)		

• **Diagram 5.42 (Steps 5-6) Swap-out—CMPLOUT Subroutine (Deciding Which Pages Are to Be Swapped Out)**



5 (Con't) If the page is part of the swap working set, fill in the SPCA entry for an "in-only" page, find the next PTE.

6 If the page is valid:

Dequeue its associated PFTE.

If an LSQA page or it is referenced or changed, fill the SPCA entry.

Find the next PTE.

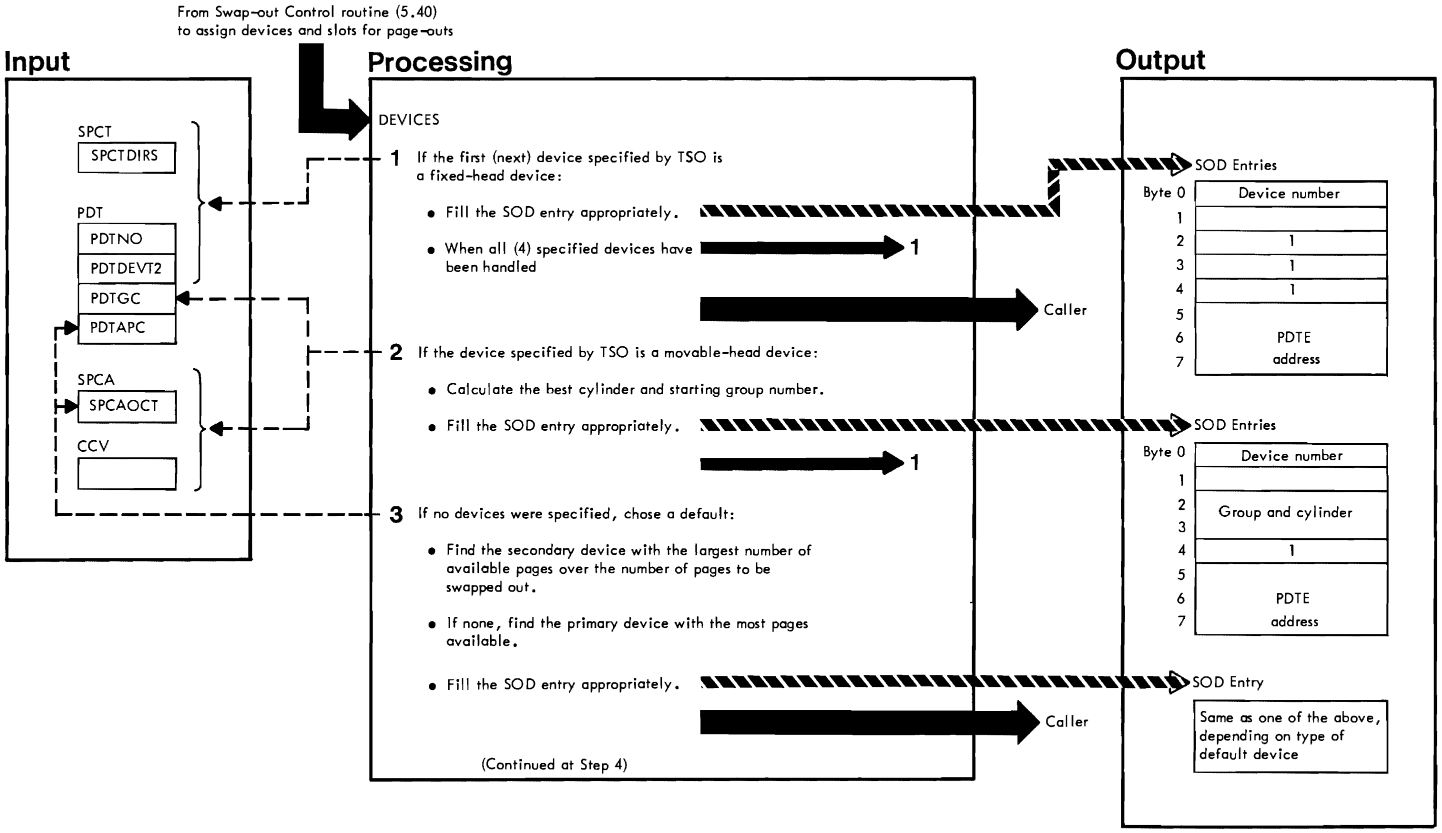
If not an LSQA page, or unreferenced and unchanged

- Route V=R intercepted page frames to a V=R region.
- Make non-V=R intercepted pages available.

• Diagram 5.42 (Steps 5-6) Swap-out - CMPLOUT Subroutine (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
5	CMPLOUT	
6 For pages that are unreferenced and unchanged and not LSQA pages: If the page is marked for V=R interception (PFTVRINT=1) it is given to V=R Release to be allocated to a pending V=R region request. Otherwise, the page is made available via PFTE Enqueue and PFTONAVQ is set to one. The PFTE is enqueued so that it will be taken first, since it cannot be reclaimed.		

Diagram 5.43 (Steps 1-3)
Swap-out — External Address Assignment Subroutines



From Swap-out Control routine (5.40) to assign devices and slots for page-outs

Input

Processing

DEVICES

1 If the first (next) device specified by TSO is a fixed-head device:

- Fill the SOD entry appropriately.
- When all (4) specified devices have been handled → 1

→ Caller

2 If the device specified by TSO is a movable-head device:

- Calculate the best cylinder and starting group number.
- Fill the SOD entry appropriately.

→ 1

3 If no devices were specified, chose a default:

- Find the secondary device with the largest number of available pages over the number of pages to be swapped out.
- If none, find the primary device with the most pages available.
- Fill the SOD entry appropriately.

→ Caller

(Continued at Step 4)

Output

SOD Entries

Byte	
0	Device number
1	
2	1
3	1
4	1
5	PDTE address
6	
7	

SOD Entries

Byte	
0	Device number
1	
2	Group and cylinder
3	
4	1
5	PDTE address
6	
7	

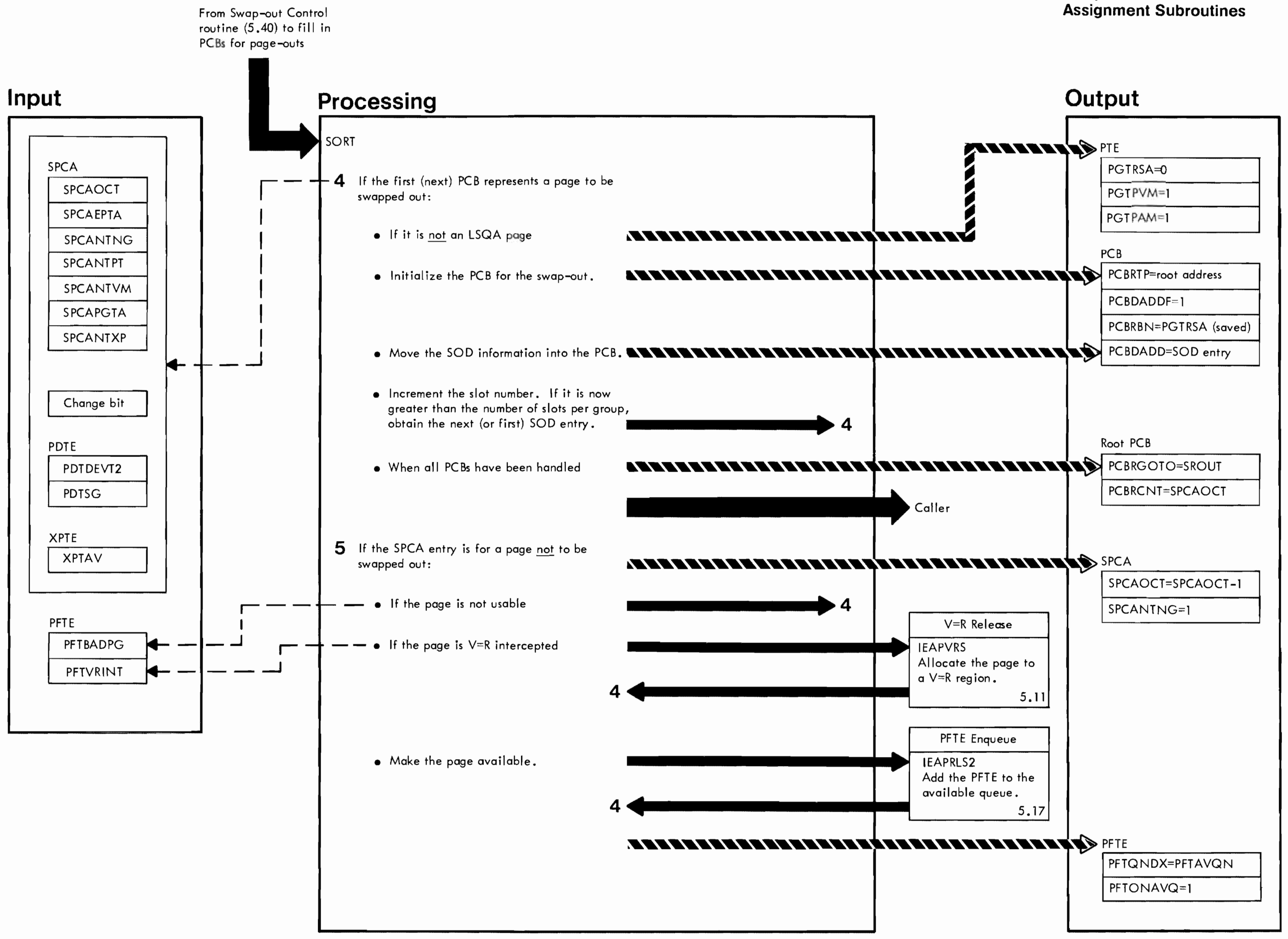
SOD Entry

Same as one of the above, depending on type of default device

• Diagram 5.43 (Steps 1-3) Swap-out - External Address Assignment Subroutines (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
* DEVICE initializes the SOD (a 32-byte internal work area) with up to 4 entries for a parallel swap or 1 entry if the swap device is defaulted.	DEVICES	
1 SPCDIRS contains indexes for 1-4 devices or a 0, indicating default.		
2 Cylinders are scanned (through counts in the CCV) for the cylinder closest to the current head location (using PDTLGN), with an available page count greater than the lesser of either the swap-out page count (SPCAOAT) or slots per cylinder - 5. If none is found, the closest cylinder with the highest available page count is selected. The end relative group number of this cylinder = cylinder number times groups per cylinder.		

Diagram 5.43 (Steps 4-5) Swap-out — External Address Assignment Subroutines



4 If the first (next) PCB represents a page to be swapped out:

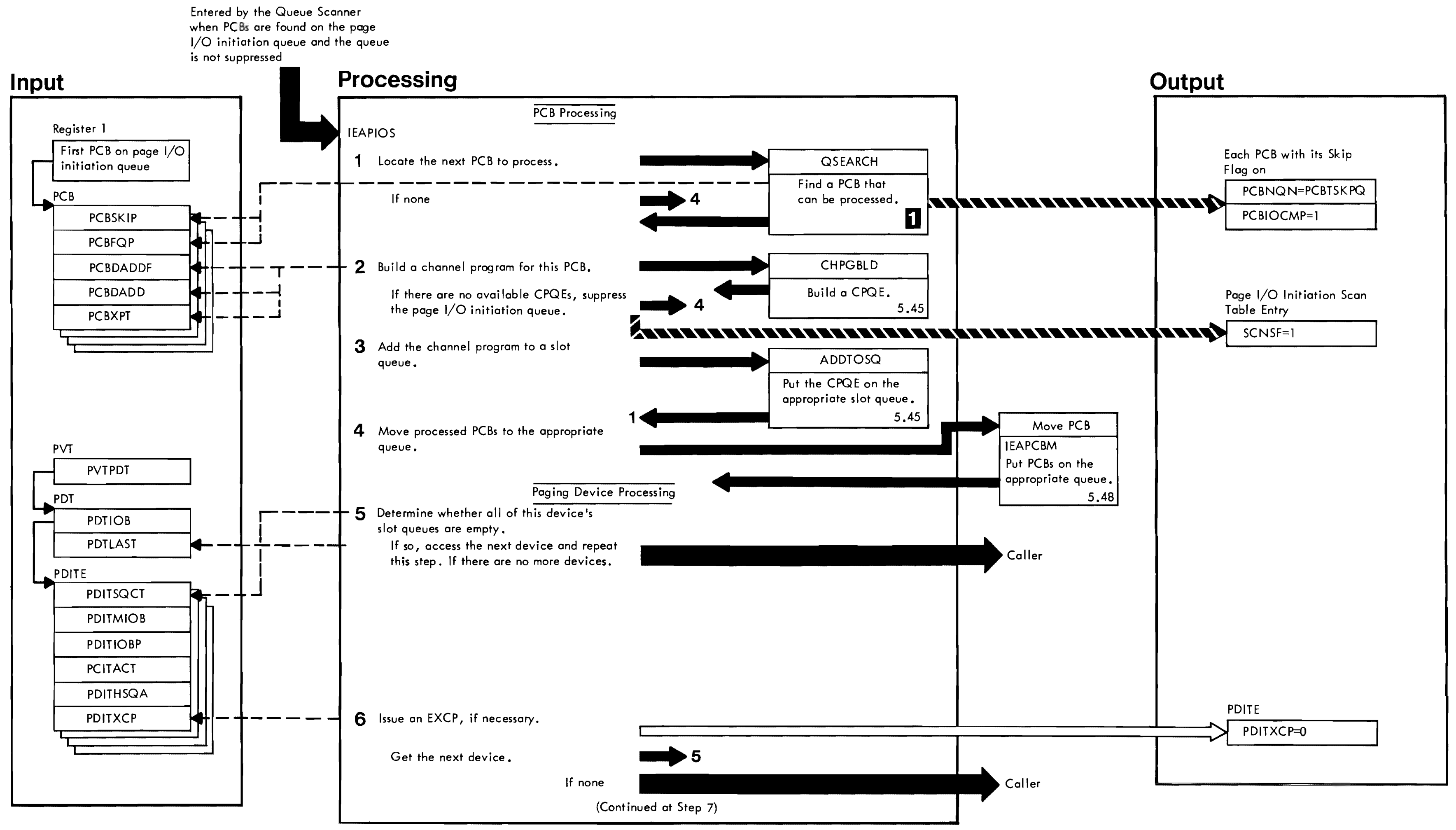
- If it is not an LSQA page
- Initialize the PCB for the swap-out.
- Move the SOD information into the PCB.
- Increment the slot number. If it is now greater than the number of slots per group, obtain the next (or first) SOD entry.
- When all PCBs have been handled

5 If the SPCA entry is for a page not to be swapped out:

- If the page is not usable
- If the page is V=R intercepted
- Make the page available.

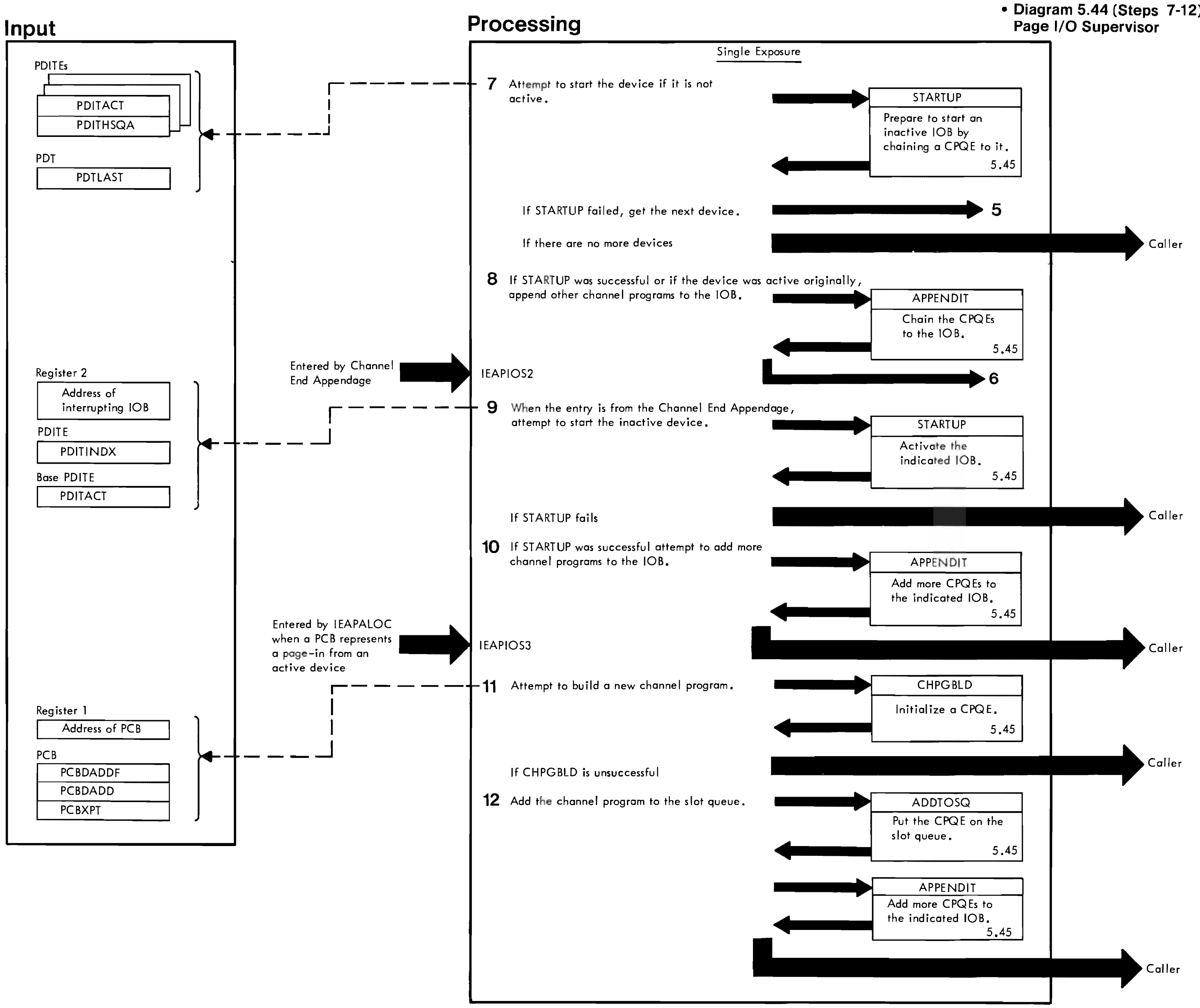
Diagram 5.43 (Steps 4-5) Swap-out - External Address Assignment Subroutines (Module IEAPSWAP)

NOTES	ROUTINE NAME	LABEL
4 The page is to be swapped out if it is not marked "in-only" (that is, if SPCANTNG=0) and: • It is an LSQA page (SPCANTPT=0) or • It is a changed page (change bit on) or • It does not have a copy on external storage (XPTAXAV=0) or • It presently exists on a movable-head device (PDTDEVT2=1).	SORT	


Entered by the Queue Scanner when PCBs are found on the page I/O initiation queue and the queue is not suppressed
Input
Register 1
First PCB on page I/O initiation queue
PCB
PCBSKIP
PCBFQP
PCBDADDF
PCBDADD
PCBXPT
PVT
PVTPDT
PDT
PDTIOB
PDTLAST
PDITE
PDITSQCT
PDITMIOB
PDITIOBP
PCITACT
PDITHSQA
PDITXCP
Processing
PCB Processing
IEAPIOS
1 Locate the next PCB to process.
If none
4
QSEARCH
Find a PCB that can be processed.
1
2 Build a channel program for this PCB.
If there are no available CPQEs, suppress the page I/O initiation queue.
4
CHPGBLD
Build a CPQE.
5.45
3 Add the channel program to a slot queue.
ADDTOSQ
Put the CPQE on the appropriate slot queue.
5.45
1
4 Move processed PCBs to the appropriate queue.
Move PCB
IEAPCBM
Put PCBs on the appropriate queue.
5.48
Paging Device Processing
5 Determine whether all of this device's slot queues are empty.
If so, access the next device and repeat this step. If there are no more devices.
Caller
6 Issue an EXCP, if necessary.
Get the next device.
5
If none
Caller
(Continued at Step 7)
Output
Each PCB with its Skip Flag on
PCBNQN=PCBTSKPQ
PCBIOCMP=1
Page I/O Initiation Scan Table Entry
SCNSF=1
PDITE
PDITXCP=0

• Diagram 5.44 (Steps 1-6) Page I/O Supervisor (Module IEAPIOS)

NOTES	ROUTINE NAME	LABEL
1	IEAPIOS	
2 The XPTE and PCB addresses are passed to CHPGBLD. If the PCB is marked for directed page-out (PCBDADDF=1), the XPTE address that is passed is actually the address of PCBDADD.		BUILD
4 This step completes PCB processing. The paging devices are handled next.		MOVIT
5		PDITLOOP
6 The EXCPs are issued by using the EXCPVR macro instruction. PDITXCP is set by STARTUP.		TESTEXCP
1 The QSEARCH subroutine searches the page I/O initiation queue for the first (next) PCB with the "skip" flag off (PCBSKIP=0). It returns the address of this PCB in register 1. If no PCBs are usable, a zero is returned. Each PCB with the "skip" flag on is marked I/O complete and is routed to the task post queue.	QSEARCH	


Input
PDITEs
PDITACT
PDITHSQA
PDT
PDTLAST
Register 2
Address of interrupting IOB
PDITE
PDITINDX
Base PDITE
PDITACT
Register 1
Address of PCB
PCB
PCBDADDF
PCBDADD
PCBXPT
Processing
Single Exposure
Entered by Channel End Appendage
IEAPIOS2
Entered by IEAPALOC when a PCB represents a page-in from an active device
IEAPIOS3
STARTUP
Prepare to start an inactive IOB by chaining a CPQE to it.
5.45
5
Caller
APPENDIT
Chain the CPQEs to the IOB.
5.45
6
STARTUP
Activate the indicated IOB.
5.45
Caller
APPENDIT
Add more CPQEs to the indicated IOB.
5.45
Caller
CHPGBLD
Initialize a CPQE.
5.45
Caller
ADDTOSQ
Put the CPQE on the slot queue.
5.45
APPENDIT
Add more CPQEs to the indicated IOB.
5.45
Caller

7 Attempt to start the device if it is not active.

If STARTUP failed, get the next device.

If there are no more devices

8 If STARTUP was successful or if the device was active originally, append other channel programs to the IOB.

9 When the entry is from the Channel End Appendage, attempt to start the inactive device.

If STARTUP fails

10 If STARTUP was successful attempt to add more channel programs to the IOB.

11 Attempt to build a new channel program.

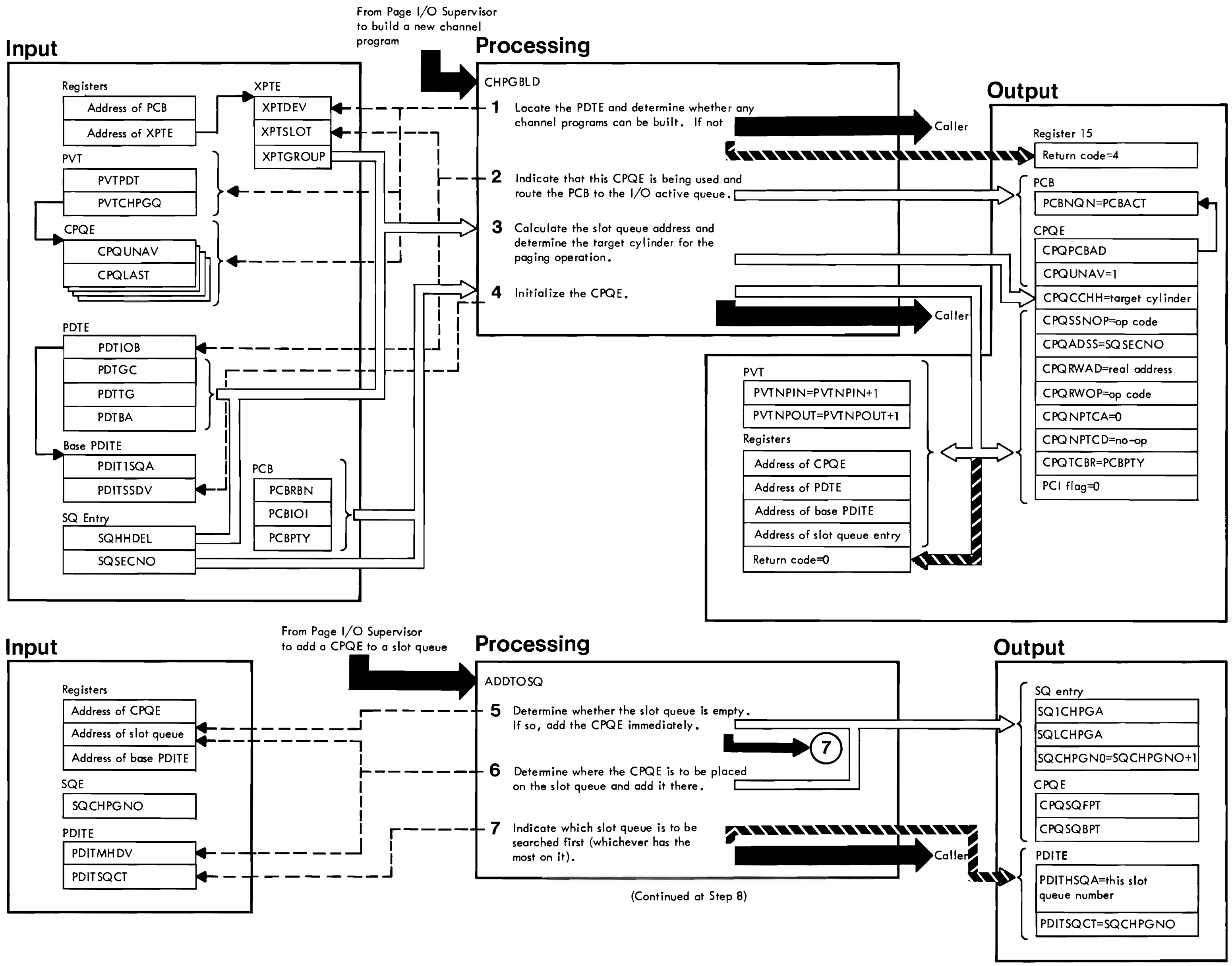
If CHPGBLD is unsuccessful

12 Add the channel program to the slot queue.

• Diagram 5.44 (Steps 7-12) Page I/O Supervisor (Module IEAPIOS)

NOTES	ROUTINE NAME	LABEL
7	IEAPIOS	MLTXP2
9 The PDITE address is the input IOB address. To minimize rotational delay, the slot queue chosen for STARTUP is the next sequential slot queue of the next sequential slot queue of the last active channel program's slot queue.	IEAPIOS2	
11 The PCB passed needs I/O initiation and the device that it requires is active. If the device is active on the same cylinder as this request, the Page I/O Supervisor calls APPENDIT. The XPTE is located as in IEAPIOS, Note 2.	IEAPIOS3	

Diagram 5.45 (Steps 1-7)
Page I/O Supervisor—
Subroutines for Building and
Queuing Channel Programs



CHPGBLD

1 Locate the PDTE and determine whether any channel programs can be built. If not

2 Indicate that this CPQE is being used and route the PCB to the I/O active queue.

3 Calculate the slot queue address and determine the target cylinder for the paging operation.

4 Initialize the CPQE.

ADDTOSQ

5 Determine whether the slot queue is empty. If so, add the CPQE immediately.

6 Determine where the CPQE is to be placed on the slot queue and add it there.

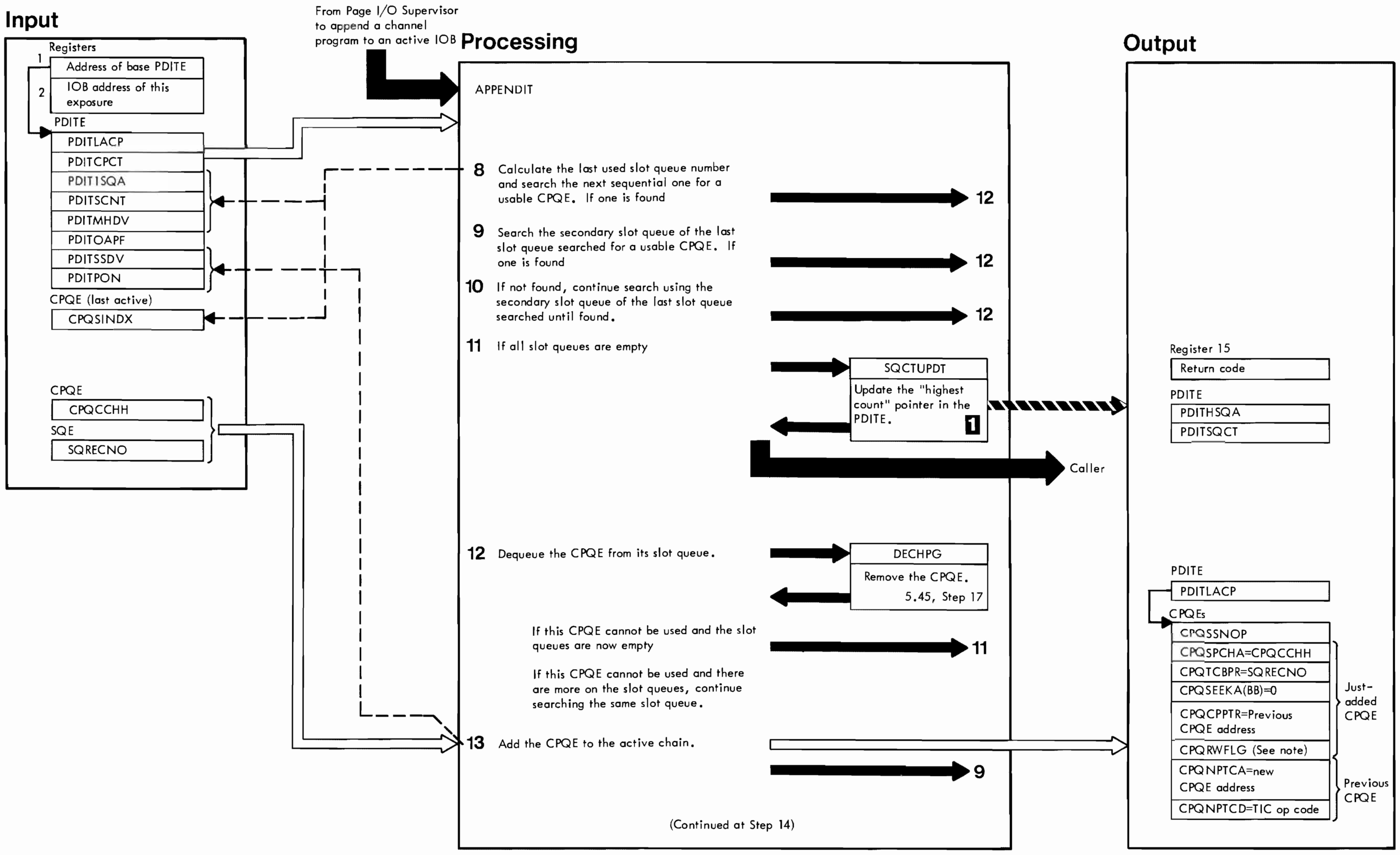
7 Indicate which slot queue is to be searched first (whichever has the most on it).

(Continued at Step 8)

• Diagram 5.45 (Steps 1-7) Page I/O Supervisor - Subroutines for Building and Queuing Channel Programs (Module IEAPIOS)

NOTES	ROUTINE NAME	LABEL
1 The PDTE address = [(XPTDEV-1)*32]+PVTPDT. The CPQEs are searched, starting from the last CPQE used, for the first available one (CPQUNAV=0). The CPQE address is incremented by 64 to find the next CPQE.	CHPGBLD	CPAVTEST
2 The correct slot queue address = PDIT1SQA+[(XPTSIOT-1)*16].		CPAVAIL
3 The target CCHH is calculated as follows: CCDELTA = $\frac{\text{XPTGROUP}}{\text{PDTGC}}$. If the remainder = 0, CCDELTA = CCDELTA-1. HHDELTA = XPTGROUP-[(CCDELTA*PDTGC)+1]*PDTTG. CCDELTA is combined with HHDELTA and the four HHDELTA bits in SQRECNO are added to the combination. The final CCHHRDELTA + PDTEA = the target CCHHR.		MULTCCD
4 When the device characteristics include rotational sensing (PDITSSDV=1), the set sector operation code and sector must be placed in the CPQE. If the PCB is for a page-in (PCBIOI=1), the read data operation code is placed in CPQRWOP and the count of page-ins (PVTNPIN) is incremented (for SMF purposes). If the PCB is for a page-out (PCBIOI=0), the write data operation code is placed in CPQRWOP and the page-out count (PVTNPOUT) is incremented.		ALLDEV
5 If the slot queue is empty (SQCHPGNO=0), none of the criteria for CPQE placement have to be considered. The first and last CPQE pointers (SQ1CHPGA and SQLCHPGA respectively) are set to the input CPQE address. The forward and backward CPQE pointers (CPQSQFPT and CPQSQBPT respectively) are set to zero. The CPQE count is incremented (SQCHPGNO = SQCHPGNO+1).	ADDTOSQ	WASEMPTY
6 For movable-head devices (PDITMHDV=1), the placement of the new CPQE on the slot queue is determined: 1. By cylinder order (low to high). 2. By priority (high to low within cylinder groups). 3. By read over write (within cylinder groups of the same priority). 4. FIFO (when the other 3 are equal). For fixed-head devices (PDITMHDV=0), only 2-4 are considered. To add the CPQE to the slot queue, the first (SQ1CHPGA) or last (SQLCHPGA) pointer is changed to the new CPQE address if necessary. The forward (CPQSQFPT) and backward (CPQSQBPT) of both this CPQE and its previous and next CPQEs are changed appropriately. The CPQE count (SQCHPGNO) is incremented by 1.		SRCHLOOP
7 If the count of CPQEs on this slot queue (SQCHPGNO) is greater than the previously high count (PDITSQCT), the "high slot queue address" (PDITHSQA) is set to this slot queue address and the high count is updated to reflect the new high (PDITSQCT=SQCHPGNO). If SQCHPGNO is smaller than PDITSQCT, no changes are made.		UPCOUNT

• **Diagram 5.45 (Steps 8-13) Page I/O Supervisor — Subroutines for Building and Queuing Channel Programs**



8 Calculate the last used slot queue number and search the next sequential one for a usable CPQE. If one is found

9 Search the secondary slot queue of the last slot queue searched for a usable CPQE. If one is found

10 If not found, continue search using the secondary slot queue of the last slot queue searched until found.

11 If all slot queues are empty

12 Dequeue the CPQE from its slot queue.

If this CPQE cannot be used and the slot queues are now empty

If this CPQE cannot be used and there are more on the slot queues, continue searching the same slot queue.

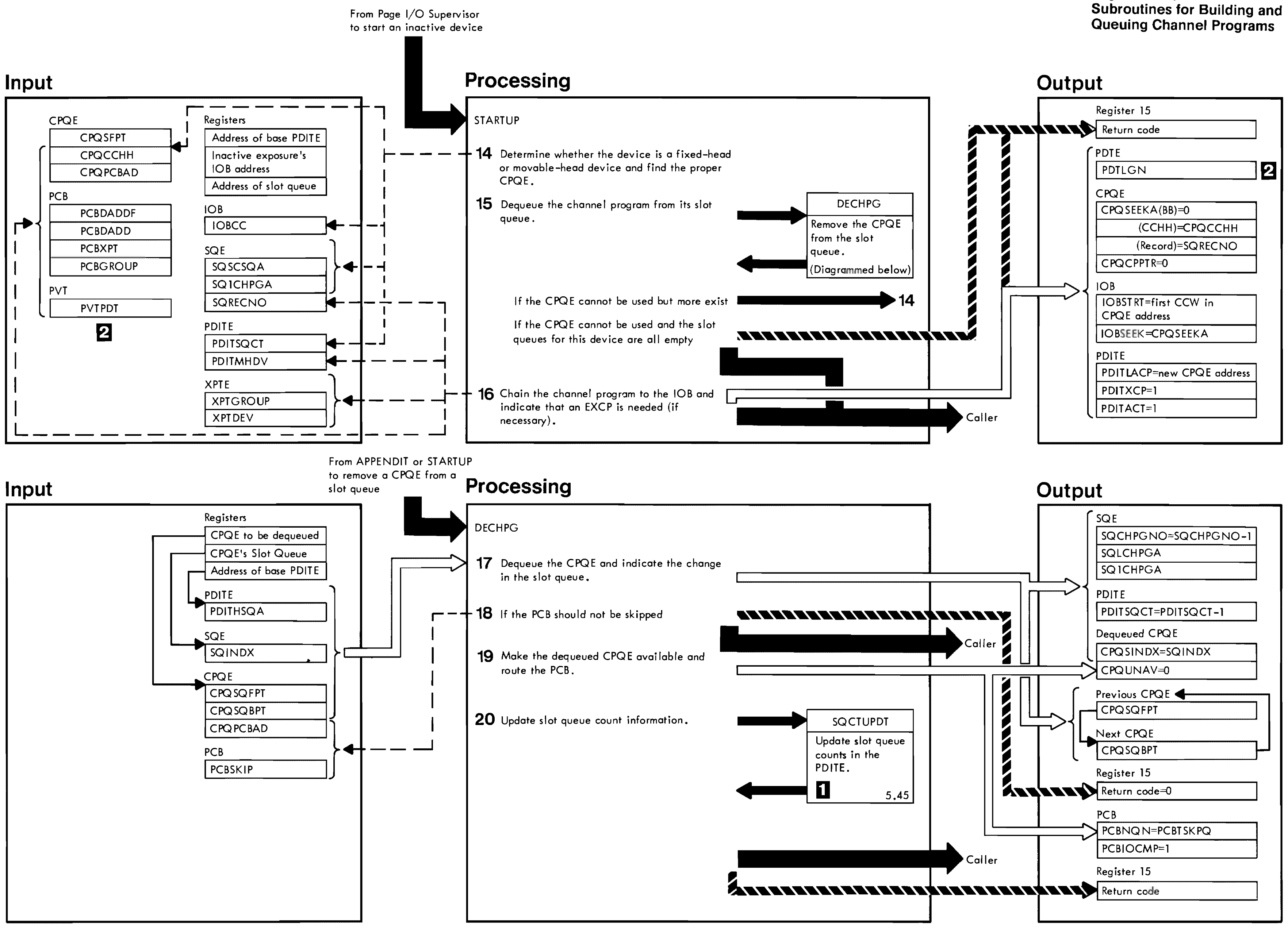
13 Add the CPQE to the active chain.

(Continued at Step 14)

• Diagram 5.45 (Steps 8-13) Page I/O Supervisor - Subroutines for Building and Queuing Channel Programs (Module IEAPICS)

NOTES	ROUTINE NAME	LABEL
8 The last-used slot queue address = (CPQSINDX*16)+ PDIT1SQA. If the device is movable head (PDITMHDV=1), the cylinder numbers of the last active channel program and the CPQE being tested must be equal in order for the CPQE to be usable. If LACP CC is higher, keep searching this slot queue. If LACP CC is lower, stop searching this slot queue.	APPENDIT	START
10 For moveable-head files, CPQEs are selected only if they are for the same cylinder.	TPRIM	
12 The new CPQE is always added to the end of the active queue.	APPENDIT	DUAPPEND
13		RC0
1 SQCTUPDT compares the count of CPQEs on each slot queue (SQCHPGNO) for this device with the high count (PDITSQCT). If a slot queue is found with a higher CPQE count, PDITSQCT is set equal to that count and PDITHSQA is set to that slot queue number. If either PDITHSQA was zero before the search, or PDITSQCT is zero at the end of the search, a return code of 7 is returned to the caller to indicate that all slot queues for this device are empty. Otherwise, a return code of 0 is given to indicate that more CPQEs exist.	SQCTUPDT	

• Diagram 5.45 (Steps 14-20) Page I/O Supervisor — Subroutines for Building and Queuing Channel Programs



STARTUP

14 Determine whether the device is a fixed-head or movable-head device and find the proper CPQE.

15 Dequeue the channel program from its slot queue.

If the CPQE cannot be used but more exist

If the CPQE cannot be used and the slot queues for this device are all empty

16 Chain the channel program to the IOB and indicate that an EXCP is needed (if necessary).

DECHPG

17 Dequeue the CPQE and indicate the change in the slot queue.

18 If the PCB should not be skipped

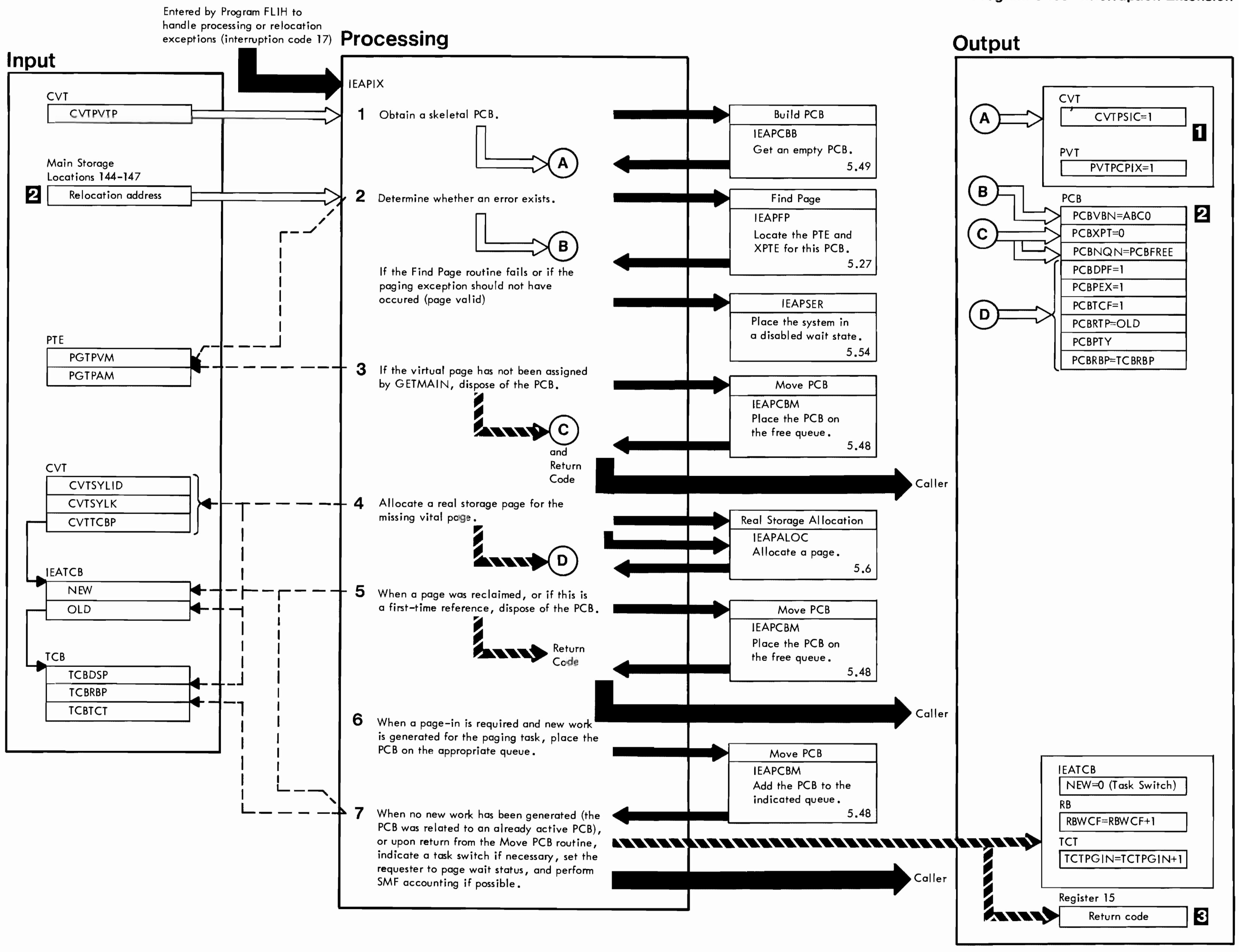
19 Make the dequeued CPQE available and route the PCB.

20 Update slot queue count information.

Diagram 5.45 (Steps 14-20) Page I/O Supervisor - Subroutines for Building and Queuing Channel Programs (Module IEAPIOS)

	NOTES	ROUTINE NAME	LABEL
14	For a movable-head device, the CPQE with the closest CC to the last cylinder used (IOBCC) is used. All slot queues are searched in secondary sequence and the CPQE with the smallest absolute difference between its CC and IOBCC is used. The search ends immediately if a match is found. For a fixed-head device, the first non-empty slot queue's first CPQE is used. The slot queues are searched in secondary sequence.	STARTUP	S010
15	A return code of 7 (which means no CPQE could be used and the slot queues are empty) is returned to the caller.		S030
16	An EXCP is needed if the caller is the Queue Scanner. A return code of 0 (a CPQE was chained) is returned to the caller.		
17	The first and last pointers of the SQE are updated if necessary and the forward and backward pointers of the previous and next CPQEs respectively are changed if necessary to reflect this CPQE's removal. If this slot queue has the highest CPQE count (PDITHSQA=this SQE), the high count (PDITSQCT) is decremented.	DECHPG	DECHPG
18	If PCBSKIP=0, this CPQE may be appended.		
19			SKIPON
20	A 0 return code from SQCTUPDT (all slot queues empty) is changed to a 4; a 7 return code (CPQEs remain on slot queues) is left alone.		
2	For a movable-head device (PDITMHDV=1), extra records must be kept for better future auxiliary storage assignment. For a directed page-out (PCBDADDF=1), the XPTE is located by PCBDADD. The PDTE address = [(XPTDEV-1)*32]+PVTPDT. PDTLGN is then set to XPTGROUP. If PCBDADDF=0, the PDTE is located by PCBDEV (rather than XPTDEV) in the above equation. PDTLGN is then set to PCBGROUP.		

• **Diagram 5.46**
Program Check Interruption Extension



Entered by Program FLIH to handle processing or relocation exceptions (interruption code 17)

IEAPIX

1 Obtain a skeletal PCB.

2 Determine whether an error exists.

If the Find Page routine fails or if the paging exception should not have occured (page valid)

3 If the virtual page has not been assigned by GETMAIN, dispose of the PCB.

4 Allocate a real storage page for the missing vital page.

5 When a page was reclaimed, or if this is a first-time reference, dispose of the PCB.

6 When a page-in is required and new work is generated for the paging task, place the PCB on the appropriate queue.

7 When no new work has been generated (the PCB was related to an already active PCB), or upon return from the Move PCB routine, indicate a task switch if necessary, set the requester to page wait status, and perform SMF accounting if possible.

Build PCB
IEAPCBB
Get an empty PCB.
5.49

Find Page
IEAPFP
Locate the PTE and XPTE for this PCB.
5.27

IEAPSER
Place the system in a disabled wait state.
5.54

Move PCB
IEAPCBM
Place the PCB on the free queue.
5.48

Real Storage Allocation
IEAPALOC
Allocate a page.
5.6

Move PCB
IEAPCBM
Place the PCB on the free queue.
5.48

Move PCB
IEAPCBM
Add the PCB to the indicated queue.
5.48

• Diagram 5.46 Program Check Interruption Extension (Module IEAPIX)

NOTES	ROUTINE NAME	LABEL
1	IEAPIX	GOPIX
2 If the return code from the Find Page routine is 4, the virtual segment in question does not exist. If it is 8, an internal error has been detected. In either case, control is passed to IEAPSER with an error code of 700. If the page is valid (PGTPVM=0), a logical error exists and control is passed to IEAPSER with a code of 701.		
3 If the virtual page is unassigned (PGTPAM=0), the XPTE pointer is zeroed and the PCB is routed to the free queue (PCBNQN=PCBFREE). Exit is made to the caller with a return code of 8 (see Note 3).		CONT10
4 If the system lock (CVTSYLK) is on and its pointer (CVTSYLID) equals OLD, the PCB is marked as representing a disabled page fault (PCBDCF=1) and the priority (PCBPTY) is set to X'FF'. Otherwise, PCBPTY=TCBDSP.		CONT20 CONT30
5 For the "reclaimed/content insensitive" case (IEAPALOC return code is zero), the PCB is disposed of (PCBNQN was set to PCBFREE by IEAPALOC). Control is returned to the caller with a return code of 4 (see Note 3).		LABEL30
6 When new work is generated (IEAPALOC return code equals either 4 or 8), the PCB representing the work is moved to the queue indicated by PCBNQN (set by IEAPALOC).		LABEL20

NOTES	ROUTINE NAME	LABEL
7 If NEW=OLD, task switch is indicated (NEW is set to zero). If TCBTCT=0, SMF accounting cannot be performed. Control is returned with a return code of 0 (see Note 3).		LABEL00 LABEL10
1 IEAPIX turns on the paging-supervisor-in-control flag (CVTPGSIC) and the "PIX in control bit" in the PVT at entry. Both are turned off before all exits.		
2 The relocation address is the virtual storage address that was found to be unavailable during dynamic address translation. Its format is <u>[00] AB] CD] EF]</u> The virtual block number is ABC0.		
3 <u>Return Code</u> / <u>Meaning</u> 0 — The current task cannot be dispatched; either the paging task will be in control or a task switch is necessary, and the current task is in page wait. 4 — The current task is dispatchable; a real storage page has been assigned and the task is ready. 8 — No page has been assigned because a logical addressing or protection exception has been detected.		

Diagram 5.47
Real to Virtual Address Translation Routine

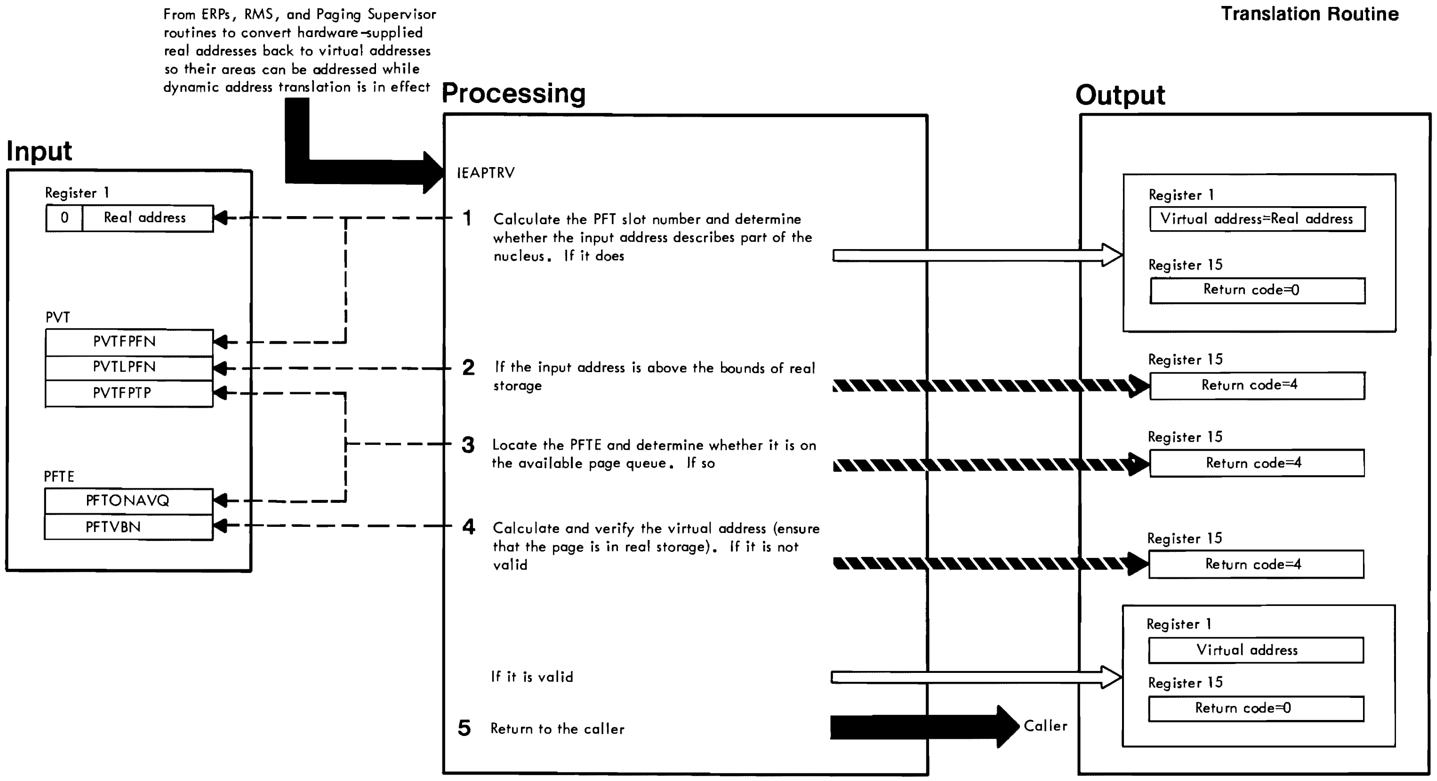
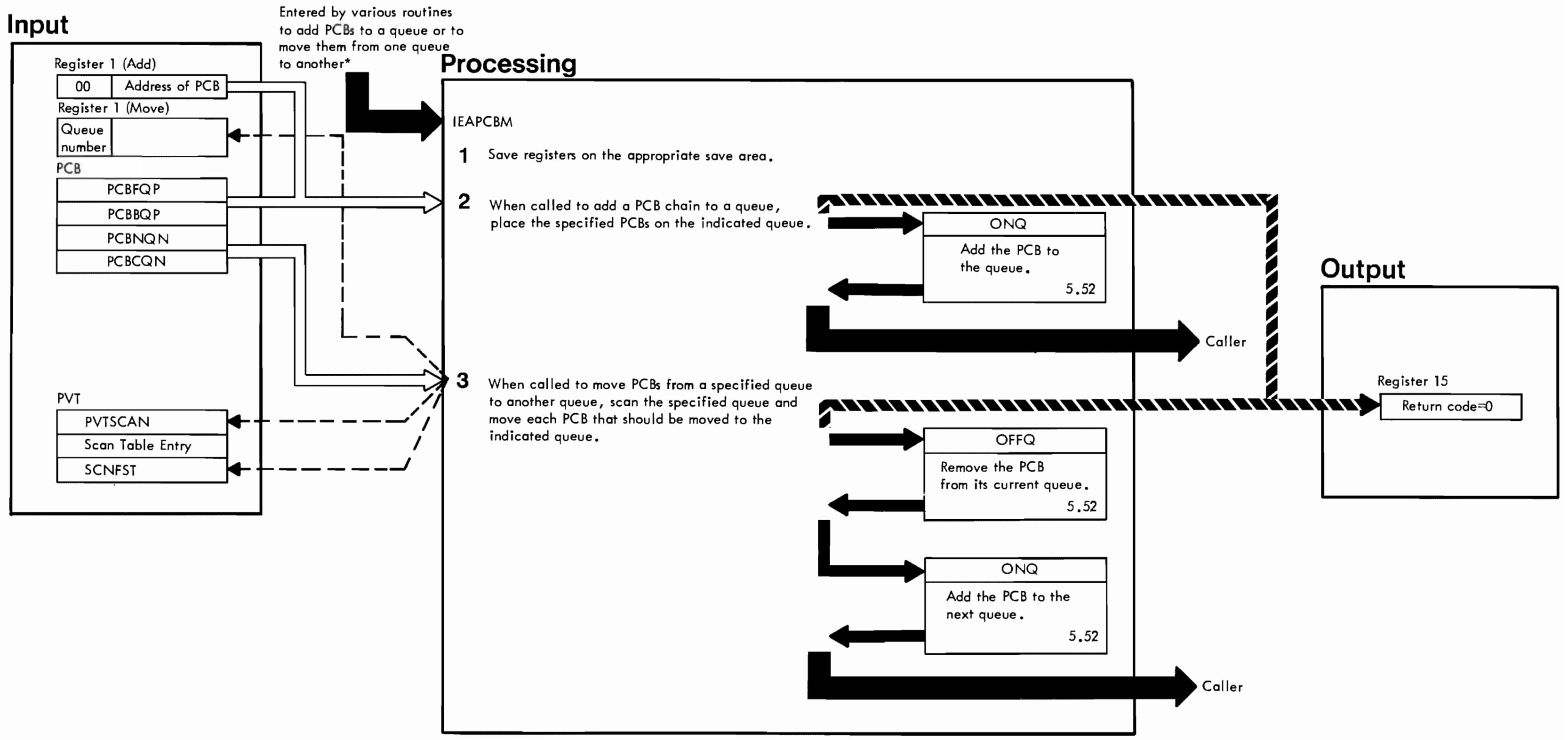

From ERPs, RMS, and Paging Supervisor routines to convert hardware-supplied real addresses back to virtual addresses so their areas can be addressed while dynamic address translation is in effect
Input
Register 1
0
Real address
PVT
PVTFPFN
PVTLPFN
PVTFPTP
PFTE
PFTONAVQ
PFTVBN
Processing
IEAPTRV
1 Calculate the PFT slot number and determine whether the input address describes part of the nucleus. If it does
2 If the input address is above the bounds of real storage
3 Locate the PFTE and determine whether it is on the available page queue. If so
4 Calculate and verify the virtual address (ensure that the page is in real storage). If it is not valid
If it is valid
5 Return to the caller
Caller
Output
Register 1
Virtual address=Real address
Register 15
Return code=0
Register 15
Return code=4
Register 15
Return code=4
Register 15
Return code=4
Register 1
Virtual address
Register 15
Return code=0

Diagram 5.47 Real to Virtual Address Translation Routine (Module IEAPTRV)

NOTES	ROUTINE NAME	LABEL
1 The PFT slot number is derived by multiplying the high order 12 bits of the input address by 16. If the PFT slot number is less than PVTFPFN, the storage it describes is in the nucleus.	IEAPTRV	GOTRV
2 If the PFT slot number is greater than PVTLPFN, the storage it refers to is above the bounds of real storage.		
3 The PFTE is located by adding the PFT slot number to the PFT origin (PFTFPTP). If PFTONAVQ=0, the address is translated.		
4 The virtual address is calculated as follows: The high-order 12 bits of PFTVBN are placed in bits 8-19 of the output address. The low-order 12 bits of the input address are placed in the low-order 12 bits of the output address. The result is checked with a Load Real Address instruction.		FINISH

Diagram 5.48
Move PCB Routine



IEAPCBM

1 Save registers on the appropriate save area.

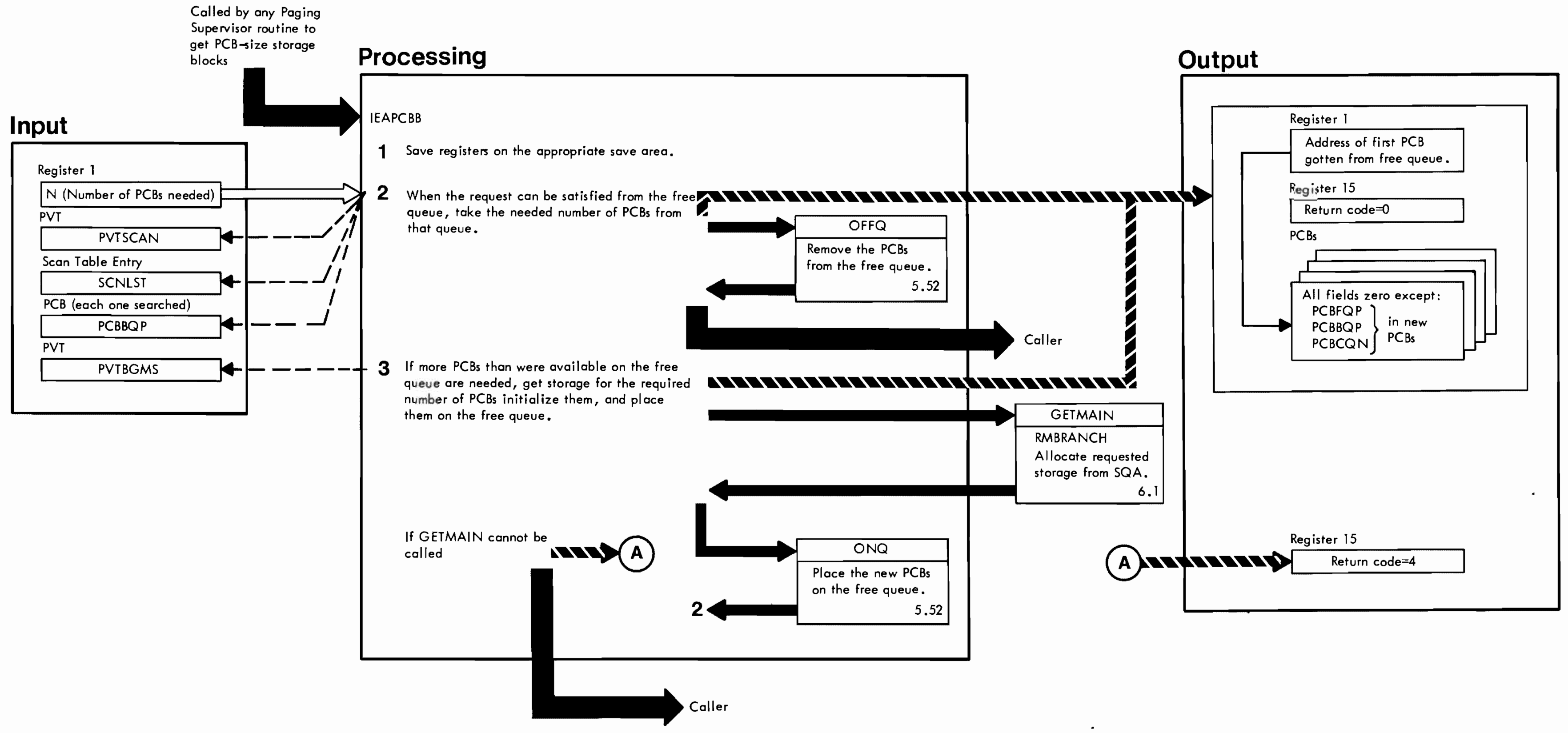
2 When called to add a PCB chain to a queue, place the specified PCBs on the indicated queue.

3 When called to move PCBs from a specified queue to another queue, scan the specified queue and move each PCB that should be moved to the indicated queue.

Diagram 5.48 Move PCB Routine (Module IEAPCB)

NOTES	ROUTINE NAME	LABEL
* Called by Program Interrupt Extension, Page Services Interface, Swap, and certain queue processors to add PCBs to a PCB queue. Called by Queue Scanner, Page I/O Post, and the Page I/O supervisor, appendages, etc., to move PCBs from one queue to another.	IEAPCBM	
1 Since Move PCB may be entered enabled or disabled, it must have two separate save areas. Upon entry, a switch is tested and if off, it is turned on and registers are saved in the disabled save area. An indicator is kept to show from which save area registers are to be restored. The switch is turned off when registers are restored from the enabled save area.		
2 Pointers to the first and last PCB on the chain are passed to ONQ.		
3 The PCB queue header for the indicated queue is calculated as follows: PVTSCAN + 12*(input queue number-1). PVTSCAN is the beginning of the scan table. 12 is the length of a scan table entry. If the PCB's next queue number (PCBNQN) is equal to the current queue number (PCBCQN), it is not moved. If they are not equal, the PCB is moved to the queue indicated by PCBNQN. OFFQ and ONQ are called for each PCB moved.		

Diagram 5.49
Build PCB Routine



Called by any Paging Supervisor routine to get PCB-size storage blocks

Input

Register 1
N (Number of PCBs needed)
PVT
PVTSCAN
Scan Table Entry
SCNLST
PCB (each one searched)
PCBBQP
PVT
PVTBGMS

Processing

IEAPCBB

1 Save registers on the appropriate save area.

2 When the request can be satisfied from the free queue, take the needed number of PCBs from that queue.

OFFQ
Remove the PCBs from the free queue.
5.52

Caller

3 If more PCBs than were available on the free queue are needed, get storage for the required number of PCBs initialize them, and place them on the free queue.

GETMAIN
RMBRANCH
Allocate requested storage from SQA.
6.1

If GETMAIN cannot be called

A

ONQ
Place the new PCBs on the free queue.
5.52

2

Caller

Output

Register 1
Address of first PCB gotten from free queue.
Register 15
Return code=0
PCBs
All fields zero except:
PCBFQP
PCBBQP
PCBCQN
in new PCBs

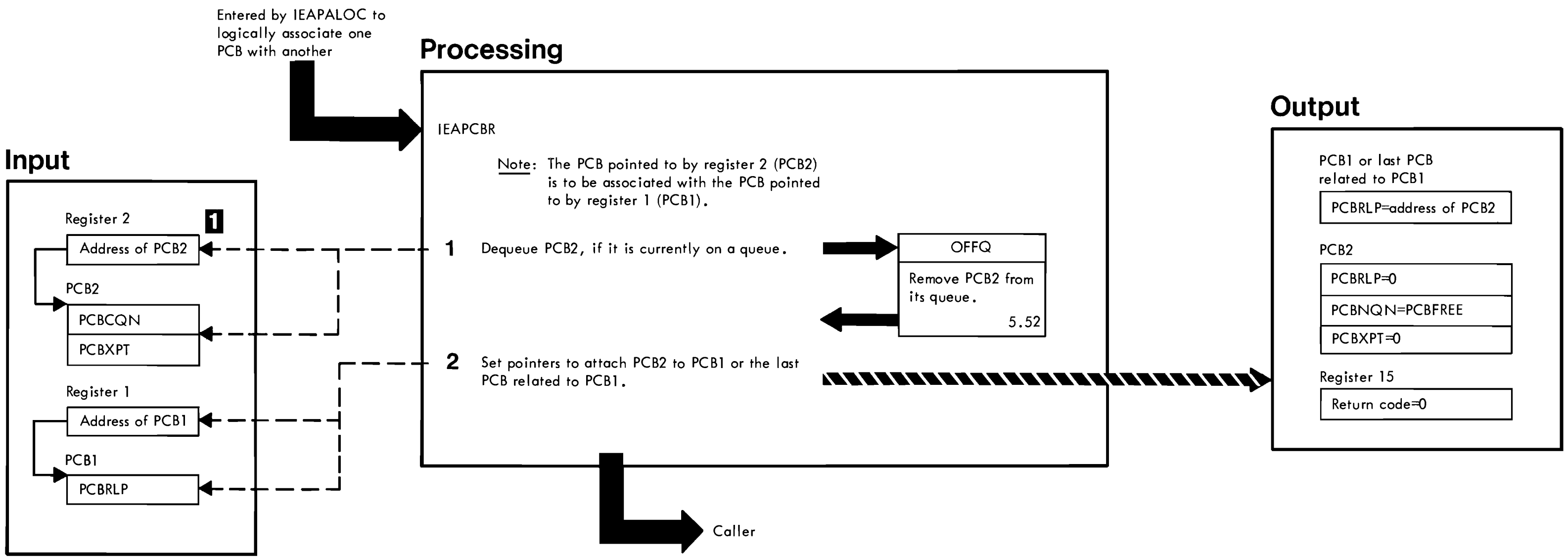
A

Register 15
Return code=4

Diagram 5.49 Build PCB Routine (Module IEAPCB)

NOTES	ROUTINE NAME	LABEL
1 Since Build PCB may be entered enabled or disabled, it must have two separate save areas. Upon entry, a switch is tested and if off, it is turned on and registers are saved in the disabled save area. An indicator is kept to show from which save area registers are to be restored. The switch is turned off when registers are restored from the enabled save area.	IEAPCBB	
2 The PCB queue header for the indicated queue is calculated as follows: PVTSCAN + 12*(input queue number-1). PVTSCAN is the beginning of the scan table. 12 is the length of a scan table entry. The free queue is scanned starting from the last PCB and following the back pointers (PCBBQP). Pointers to the Nth PCB and the last PCB on the free queue are passed to OFFQ.		
3 If the GETMAIN recursion switch is on (PVTBGMS=1), GETMAIN cannot be called to get the storage required and control is returned to the caller. Otherwise, GETMAIN is requested to allocate enough storage for N-(number of PCBs on the free queue) PCBs. Pointers to the beginning and end of the newly initialized PCB chain are passed to ONQ.		

• **Diagram 5.50**
Relate PCB Routine



Entered by IEAPALOC to logically associate one PCB with another
Input
Register 2
1
Address of PCB2
PCB2
PCBCQN
PCBXPT
Register 1
Address of PCB1
PCB1
PCBRLP
Processing
IEAPCBR
Note: The PCB pointed to by register 2 (PCB2) is to be associated with the PCB pointed to by register 1 (PCB1).
1 Dequeue PCB2, if it is currently on a queue.
OFFQ
Remove PCB2 from its queue.
5.52
2 Set pointers to attach PCB2 to PCB1 or the last PCB related to PCB1.
Caller
Output
PCB1 or last PCB related to PCB1
PCBRLP=address of PCB2
PCB2
PCBRLP=0
PCBNQN=PCBFREE
PCBXPT=0
Register 15
Return code=0

Diagram 5.50 Relate PCB Routine (Module IEAPCB)

NOTES	ROUTINE NAME	LABEL
1 PCB1 is a PCB for which an explicit paging operation is currently in progress. PCB2 is to be added to PCB1's related queue.	IEAPCBR	
1 PCBXPT is saved and zeroed before calling OFFQ and is restored upon return.		
2 If PCB1's related queue pointer (PCBRLP) is zero, it is set to the address of PCB2. Otherwise, PCB1's related chain is traced until the last PCB is found (PCBRLP=0) and its pointer is set to the address of PCB2.		

Diagram 5.51
Release Queue Suppression Routine

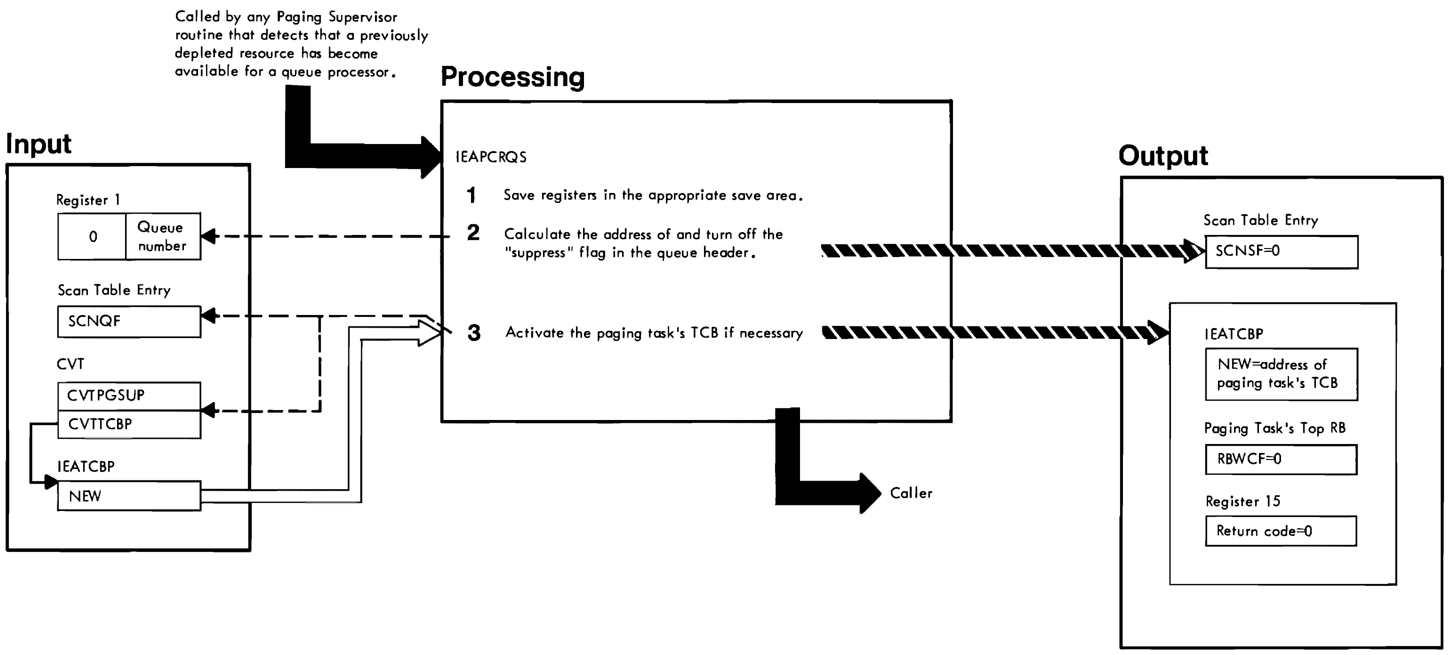
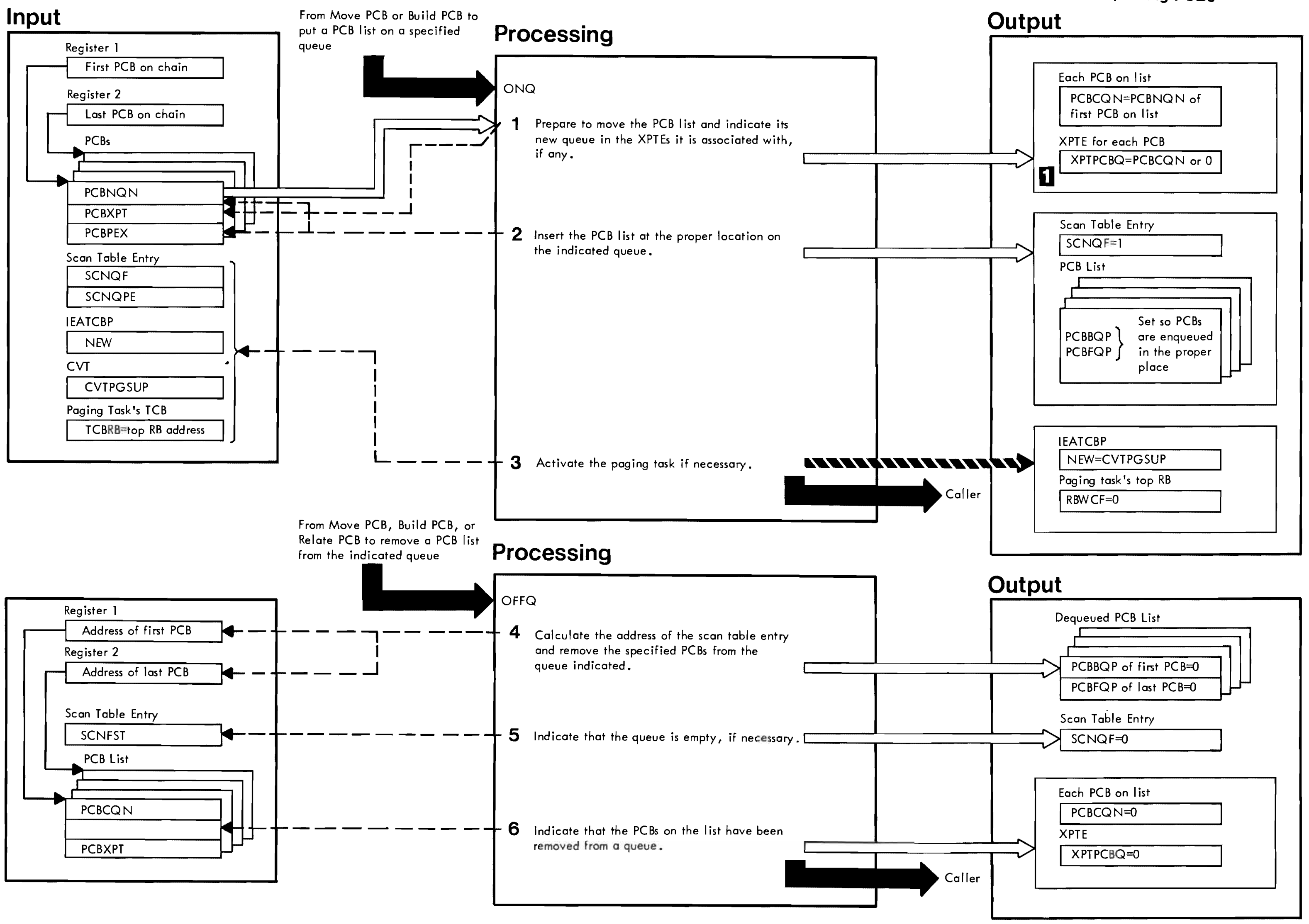

Called by any Paging Supervisor routine that detects that a previously depleted resource has become available for a queue processor.
Input
Register 1
0
Queue number
Scan Table Entry
SCNQF
CVT
CVTPGSUP
CVTTCBP
IEATCBP
NEW
Processing
IEAPCRQS
1 Save registers in the appropriate save area.
2 Calculate the address of and turn off the "suppress" flag in the queue header.
3 Activate the paging task's TCB if necessary
Caller
Output
Scan Table Entry
SCNSF=0
IEATCBP
NEW=address of paging task's TCB
Paging Task's Top RB
RBWCF=0
Register 15
Return code=0

Diagram 5.51 Release Queue Suppression Routine (Module IEAPCB)

NOTES	ROUTINE NAME	LABEL
1 Since Relate PCB may be entered enabled or disabled, it must have two separate save areas. Upon entry, a switch is tested and if off, it is turned on and registers are saved in the disabled save area. An indicator is kept to show from which save area registers are to be restored. The switch is turned off when registers are restored from the enabled save area.	IEAPCRQS	
2 The PCB queue header for the indicated queue is calculated as follows: PVTSCAN + 12*(input queue number-1). PVTSCAN is the beginning of the scan table. 12 is the length of a scan table entry.		
3 If the "NEW" TCB pointer does not equal the paging task's TCB pointer (CVTPGSUP), "NEW" is set to CVTPGSUP. If, in addition, SCNQF in this queue's header =1 (there are TCB's on the queue), the RB wait count (RBWCV) is zeroed.		

Diagram 5.52 Move/Build/Relate PCB—Subroutines for Queuing and Dequeuing PCBs



From Move PCB or Build PCB to put a PCB list on a specified queue

1 Prepare to move the PCB list and indicate its new queue in the XPTEs it is associated with, if any.

2 Insert the PCB list at the proper location on the indicated queue.

3 Activate the paging task if necessary.

From Move PCB, Build PCB, or Relate PCB to remove a PCB list from the indicated queue

4 Calculate the address of the scan table entry and remove the specified PCBs from the queue indicated.

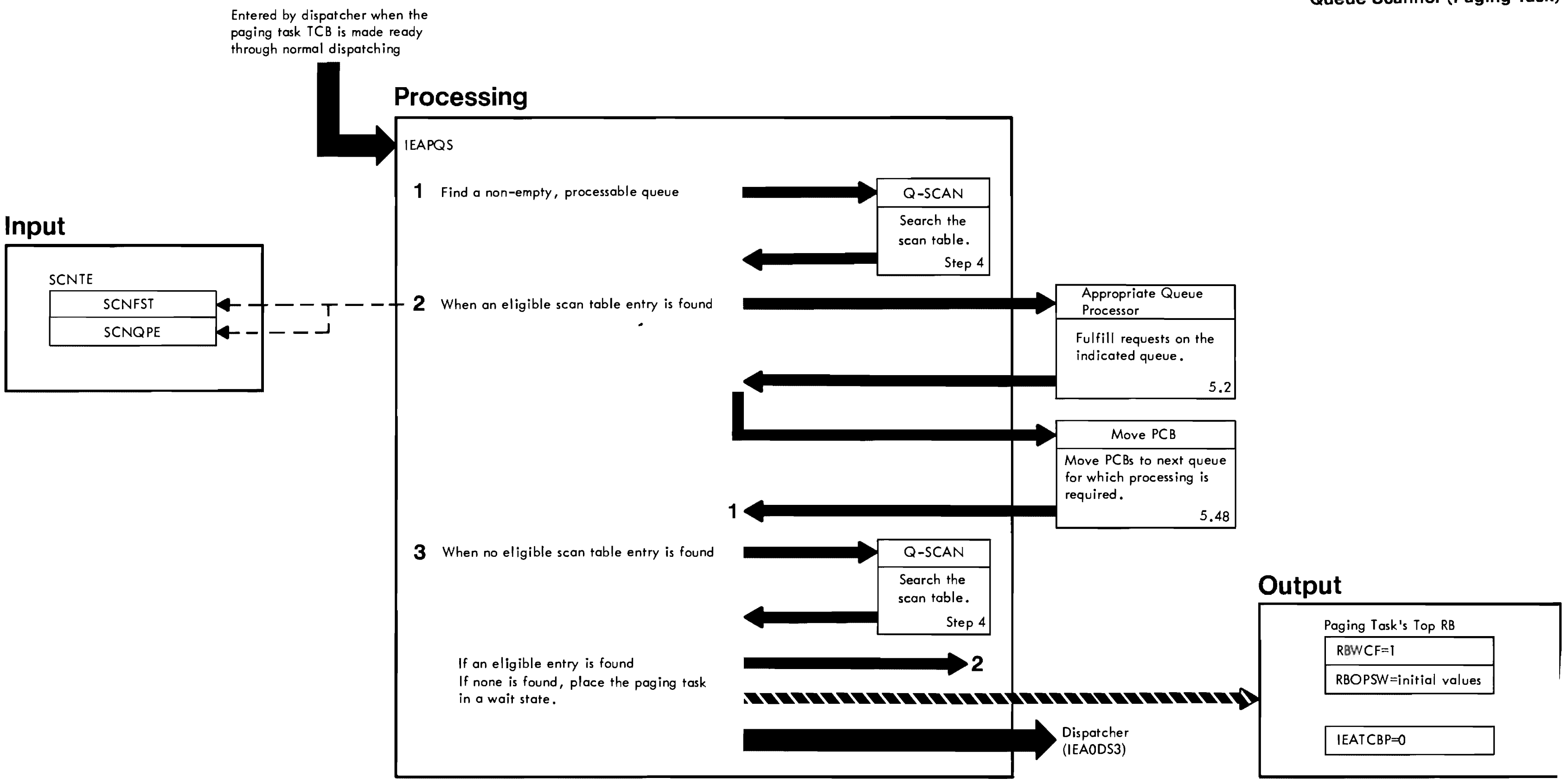
5 Indicate that the queue is empty, if necessary.

6 Indicate that the PCBs on the list have been removed from a queue.

• Diagram 5.52 Move/Build/Relate PCB - Subroutines for Queuing and Dequeuing PCBs (Module IEAPCB)

NOTES	ROUTINE NAME	LABEL
1 If the current queue number (PCBCQN) points to the free queue and PCBXPT≠0, XPTPCBQ is set to zero to indicate that no "in-process" PCB exists for this page. Otherwise, CPTPCBQ is set to PCBCQN.	ONQ	
2 The next queue number (PCBNQN) of the first PCB is tested and: If it is zero, return is made to the caller. If it points to the free queue, the PCB list is inserted at the top of the free queue. If it points to the page I/O initiation queue and: Page exception is indicated (PCBPEX=1), the PCB list is inserted at the top of the page I/O queue. Page exception is not indicated (PCBPEX=0), the PCB list is inserted at the bottom of the page I/O queue. If it points to any other queue, the PCB list is inserted at the bottom of the indicated queue.		
3 The paging task is activated ("NEW" is set equal to CVTPGSUP and RBWCF is zeroed) if: The queue added to is not suppressed (SCNSF≠0), and The queue added to is dispatchable (SCNQPE≠0), and "NEW"≠CVTPGSUP, and It is not already active.		
4 The queue header is located as in IEAPCBM, Note 3.	OFFQ	
5 The queue is empty if the first PCB pointer (SCNFST) equals zero.		
6 If PCBNQN≠PCBFREE and PCBXPT points to an XPTE, XPTPCBQ is zeroed.		

Diagram 5.53
Queue Scanner (Paging Task)


Entered by dispatcher when the paging task TCB is made ready through normal dispatching
Input
SCNTE
SCNFST
SCNQPE
Processing
IEAPQS
1 Find a non-empty, processable queue
Q-SCAN
Search the scan table.
Step 4
2 When an eligible scan table entry is found
Appropriate Queue Processor
Fulfill requests on the indicated queue.
5.2
Move PCB
Move PCBs to next queue for which processing is required.
5.48
1
3 When no eligible scan table entry is found
Q-SCAN
Search the scan table.
Step 4
If an eligible entry is found
2
If none is found, place the paging task in a wait state.
Dispatcher (IEAODS3)
Output
Paging Task's Top RB
RBWCF=1
RBOPSW=initial values
IEATCBP=0

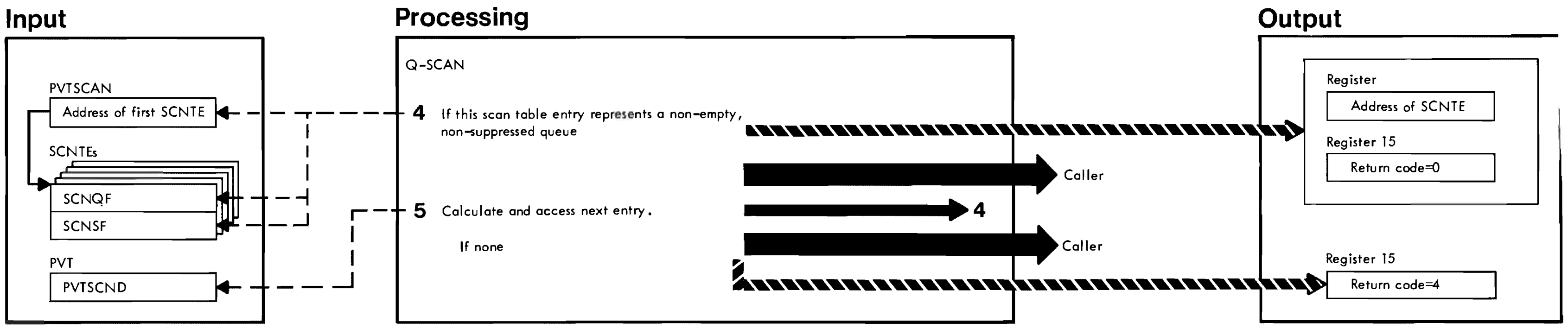


Diagram 5.53 Queue Scanner (Paging Task) (Module IEAPQS)

NOTES	ROUTINE NAME	LABEL
1	IEAPQS	QBEGNSCN
2 IEAPQS directs the flow of control to the paging supervision processor associated with the processable queues containing pending requests. The queue is flagged "in-process" (SCNQIPF=1) before the call to the processor and is marked "not-in-process" (SCNQIPF=0) upon return.		QPROCESS
3 The routine is disabled during this call to QSCAN. Since no more work exists for the paging supervisor, it puts itself in a wait state.		QLASTTRY
4 The "Q" flag must be on and the "suppress" flag must be off (SCNQF=1 and SCNQS=0).	QSCAN	QSCAN
5 The next entry is located 12 bytes past the last entry.		QTESTNEW
Note: The SCNTE of the first PCB queue is scanned first at each entry to QSCAN.		

Diagram 5.54
Paging Supervisor Error Recorder

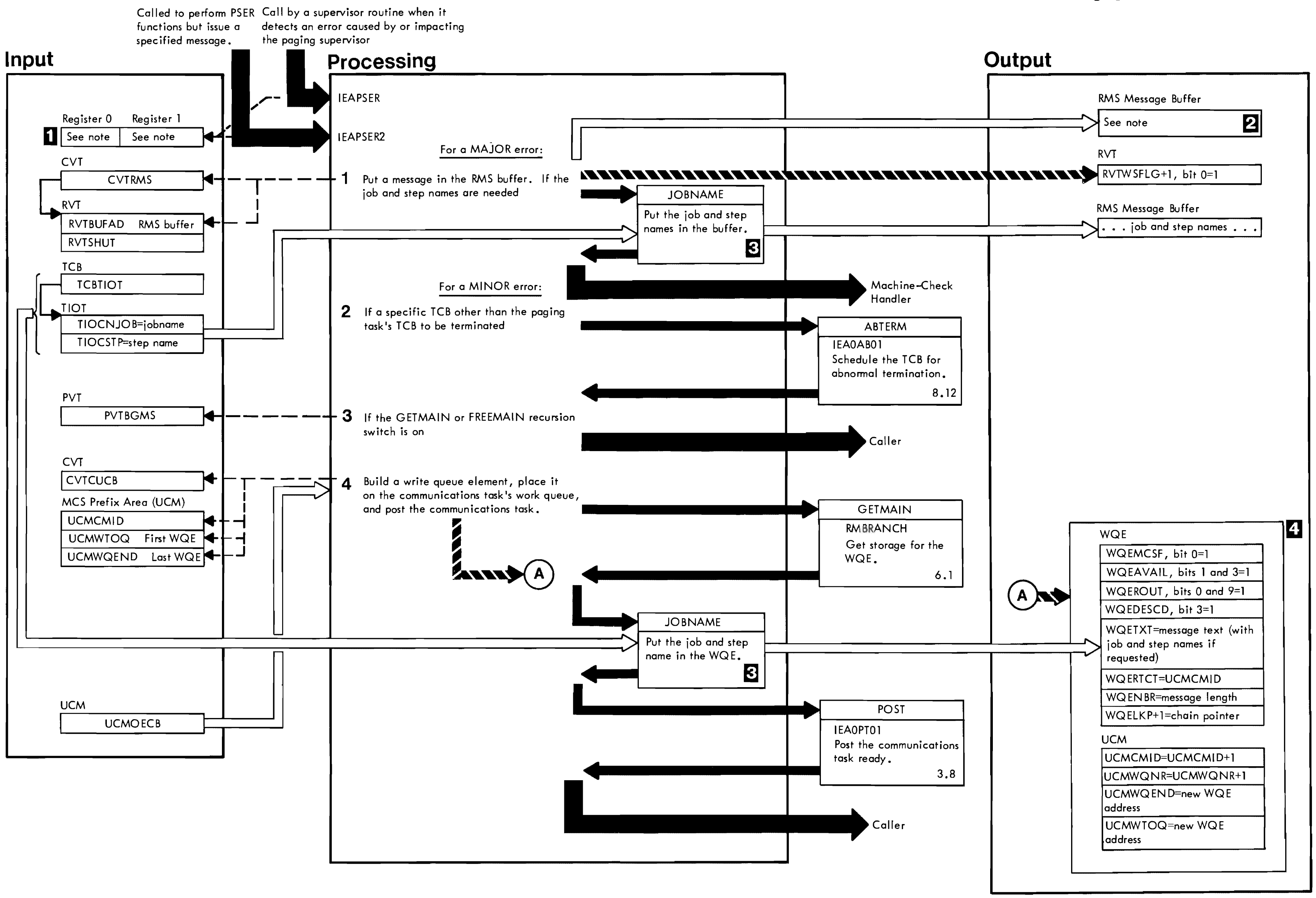
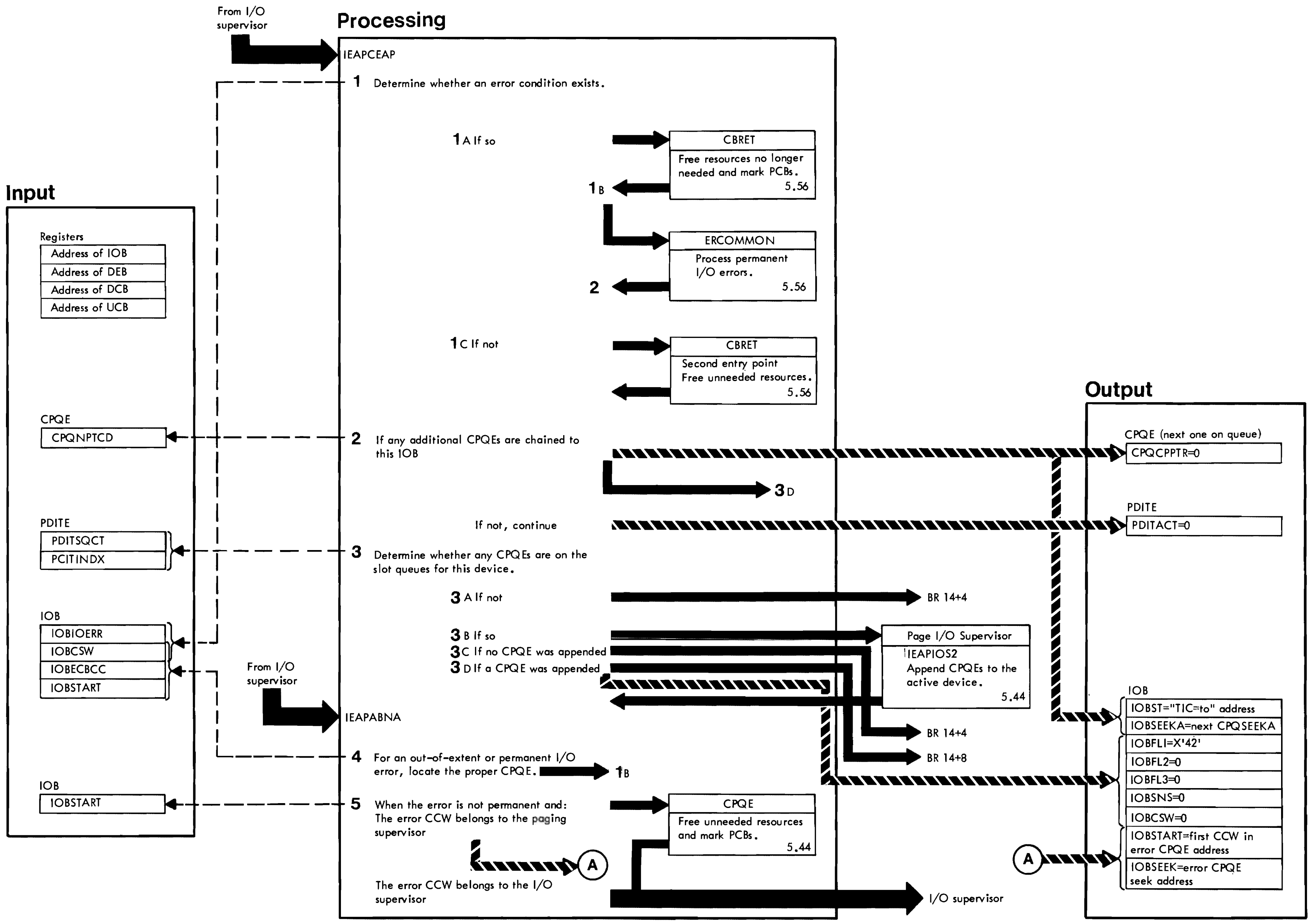

Called to perform PSER functions but issue a specified message.
Call by a supervisor routine when it detects an error caused by or impacting the paging supervisor
Input
Processing
Output
Register 0
Register 1
1
See note
See note
CVT
CVTRMS
RVT
RVTBUFAD RMS buffer
RVTSHUT
TCB
TCBTIOT
TIOT
TIOCNJOB=jobname
TIOCSTP=step name
PVT
PVTBGMS
CVT
CVTCUCB
MCS Prefix Area (UCM)
UCMCMID
UCMWTOQ First WQE
UCMWQEND Last WQE
UCM
UCMOECB
IEAPSER
IEAPSER2
For a MAJOR error:
1 Put a message in the RMS buffer. If the job and step names are needed
JOBNAME
Put the job and step names in the buffer.
3
For a MINOR error:
2 If a specific TCB other than the paging task's TCB to be terminated
Machine-Check Handler
ABTERM
IEA0AB01
Schedule the TCB for abnormal termination.
8.12
3 If the GETMAIN or FREEMAIN recursion switch is on
Caller
4 Build a write queue element, place it on the communications task's work queue, and post the communications task.
A
GETMAIN
RMBRANCH
Get storage for the WQE.
6.1
JOBNAME
Put the job and step name in the WQE.
3
POST
IEA0PT01
Post the communications task ready.
3.8
Caller
RMS Message Buffer
See note
2
RVT
RVTWSFLG+1, bit 0=1
RMS Message Buffer
. . . job and step names . . .
4
WQE
WQEMCSF, bit 0=1
WQEAVAIL, bits 1 and 3=1
WQEROUT, bits 0 and 9=1
WQEDESCD, bit 3=1
WQETXT=message text (with job and step names if requested)
WQERTCT=UCMCMID
WQENBR=message length
WQELKP+1=chain pointer
UCM
UCMCMID=UCMCMID+1
UCMWQNR=UCMWQNR+1
UCMWQEND=new WQE address
UCMWTOQ=new WQE address
A

Diagram 5.54 Paging Supervisor Error Recorder (Module IEAPSER)

NOTES	ROUTINE NAME	LABEL
1 If the MAJOR option is chosen, exit is taken to the RMS Message Writer/Wait State Effector (MCH) to write a message to the operator console and place the system in a disabled wait state.	IEAPSSR and IEAPSER2	MCHEXIT
2 This is done only if the ABTERM option was specified.		
3 If the GETMAIN/FREEMAIN recursion switch is on (PVTBGMS=1), GETMAIN cannot be called to get the storage for the WQE so control is returned to the caller. No message is printed in this case.		GMCHK
4 The WQE represents a WTO operation to the communications task.		
1 When entry is at IEAPSER, register contents are: Register 0: Byte 0 - Ignored Byte 1 - Bits 0-5 - Must be zero Bit 6 - ABTERM option flag ON= Terminate the designated task OFF= Do not terminate any task Bit 7 - MAJOR/MINOR option flag ON= MINOR error OFF= MAJOR error Bytes 2-3 - Error number (indicates caller and error) Register 1: 0 - If the ABTERM option flag is on, use the TCB address in "OLD" (IEATCBP+4) to schedule abnormal termination. TCB address - If the ABTERM option flag is on, use this TCB address to schedule abnormal termination.	IEAPSER	
When entry is at IEAPSER2, the register contents are: Register 0: Byte 0 - Same as Byte 1 above Bytes 1-3 - Address of the message to be written. Register 1: If the ABTERM option is selected or if the message contains a jobname/stepname field: =0, "OLD" TCB is to be used; or =TCB address of TCB to be used. The message pointed to in register 0 must be of that format: D L C TEXT -2 -1 -0 1 D: The displacement into the message at which the jobname and stepname of the task whose TCB address is indicated by register 1 is to be placed. If D=0, the jobname and stepname will not be inserted. L: The message length (including C). Up to 50 bytes is allowed if the MAJOR option is specified; 255 bytes are allowed for the MINOR option. C: The first character of the message. * if the message requires operator action; blank otherwise. TEXT: The actual message text beginning with IEANNNX where NNN is a unique message number and X indicates the message type.	IEAPSER2	

NOTES	ROUTINE NAME	LABEL
2 If a message was not supplied (entry at IEAPSER), the buffer content at exit is: Wait State Code X'0028' 49 *IEA047A↑ PG.SUP.ERROR ↑↑NNNN↑↑↑ SYSTEM↑TERMINATED If a message was supplied (entry at IEAPSER2), the buffer content at exit is: Wait State Message Supplied message text (with Code X'0028' Length jobname and stepname if indicated) 0 2 4		
3 The JOBNAMF subroutine is passed the address into which the character string 'jjjjjjjj.ssssssss' is to be placed. 'jjjjjjjj' is the jobname and 'ssssssss' is the stepname	JOBNAME	
4 The WQE is set to indicate the following: WQEMCSF (MCS flags): Bit 0 - Routing or descriptor codes exist. WQEAVAIL: Bit 1 - Buffer in use. Bit 3 - Buffer obtained dynamically. WQEROUT (Routing Codes): Bit 0 - Master Console. Bit 9 - System Error Maintenance. WQEDESCD (Descriptor Codes): Bit 3 - System Status. WQERTCT: Routed WQE count. WQENBR: Message length. WQETXT: If no message was supplied and ABTERM was not called: ↑IEA048I↑ PG.SUP.ERROR ↑↑NNNN↑↑↑RECOVERY↑ATTEMPTED	IEAPSER and IEAPSER2	GMRETURN
If no message was supplied and ABTERM was called: ↑IEA049I↑ PG.SUP.ERROR ↑↑NNNN↑↑↑JOBSTEP=jjjjjjjj. ssssssss↑TERMINATING		CANEDMSG
If a message was supplied it is placed in this field with the jobname and stepname if indicated.		USERMSG

• Diagram 5.55
Paging Supervisor Appendages
(Channel End, Abnormal End)

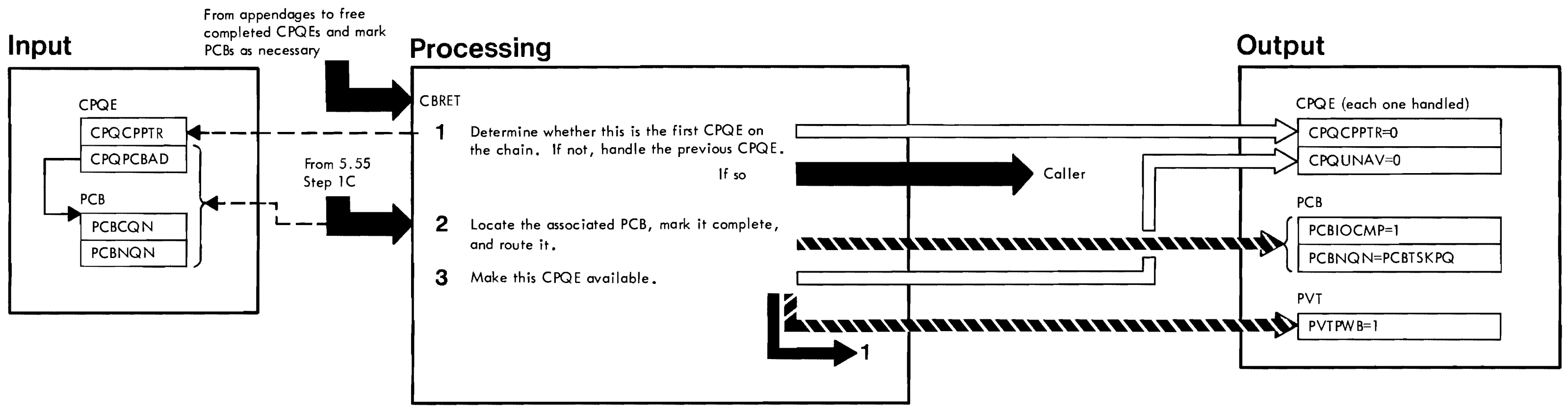

Input
Registers
Address of IOB
Address of DEB
Address of DCB
Address of UCB
CPQE
CPQNPTCD
PDITE
PDITSQCT
PCITINDX
IOB
IOBIOERR
IOBCSW
IOBECBCC
IOBSTART
IOB
IOBSTART
From I/O supervisor
Processing
IEAPCEAP
1 Determine whether an error condition exists.
1A If so
CBRET
Free resources no longer needed and mark PCBs.
5.56
1B
ERCOMMON
Process permanent I/O errors.
5.56
2
1C If not
CBRET
Second entry point
Free unneeded resources.
5.56
2 If any additional CPQEs are chained to this IOB
3D
If not, continue
3 Determine whether any CPQEs are on the slot queues for this device.
3A If not
BR 14+4
3B If so
3C If no CPQE was appended
3D If a CPQE was appended
Page I/O Supervisor
IEAPIOS2
Append CPQEs to the active device.
5.44
BR 14+4
BR 14+8
From I/O supervisor
IEAPABNA
4 For an out-of-extent or permanent I/O error, locate the proper CPQE.
1B
5 When the error is not permanent and:
The error CCW belongs to the paging supervisor
CPQE
Free unneeded resources and mark PCBs.
5.44
A
The error CCW belongs to the I/O supervisor
I/O supervisor
Output
CPQE (next one on queue)
CPQCPPTR=0
PDITE
PDITACT=0
IOB
IOBST="TIC=to" address
IOBSEEKA=next CPQSEEKA
IOBFL1=X'42'
IOBFL2=0
IOBFL3=0
IOBSNS=0
IOBCSW=0
IOBSTART=first CCW in error CPQE address
IOBSEEK=error CPQE seek address
A

• Diagram 5.55 Paging Supervisor Appendages (Channel End, Abnormal End) (Module IEAPPCIA)

NOTES	ROUTINE NAME	LABEL
* The Channel End Appendage gets control when one of the following I/O interruption terminates a paging I/O operation without any other abnormal conditions: • Channel end. • Unit exception (with or without channel end). • Channel end with wrong-length exception.	IEAPCEAP	IEAPCEAP
1		CE01 TOERCOM
2 The CPQE and PDITE are located as above.		CE03
More work will exist (CPQNPTCD=TIC operation code) when a CPQE has been appended but the Channel End Interruption was presented before the channel was aware of the new CPQE.		CE02
3 The return to 4 past register 14 will cause IOS to free the RQE but bypass posting the completion of the I/O event.		CERET01
The return to 8 past register 14 will cause IOS to return the RQE to its logical channel queue and bypass posting.		CE04

NOTES	ROUTINE NAME	LABEL
** The Abnormal End Appendage is entered whenever one of the following conditions arises: • Out-of-extent errors (probably an error in IEAPIOS or one of its control blocks). • SIO errors. • The CSW stored for an I/O interruption indicates an error other than one for which the Channel End Appendage gets control.	IEAPABNU	IEAPABNU
4 There are two classes of I/O errors -- nonpermanent and permanent. In most cases it is the Error Recovery Programs (ERPs) that classify an error as permanent. When an error is first detected, the Abnormal End Appendage will get control before the ERPs. Thus, the error will not be marked as permanent. If the Abnormal End Appendage returns to IOS at the address in register 14, the ERPs will gain control. For errors from which recovery is impossible, the ERPs will mark the error as permanent immediately. In this case, the Abnormal End Appendage will be reentered immediately with the error marked as permanent. In cases where the I/O operation may succeed if tried again, the ERPs will retry the failing I/O operation until it succeeds (in which case the Abnormal End Appendage will not be reentered) or until it has failed a specified number of times (in which case the Abnormal End Appendage is reentered with the error marked as permanent.		AE020
5 For an out-of-extent or permanent error in a non-paging supervisor CCW, use the CPQE from IOBSTART.		
For a permanent error in a paging supervisor CCW, use the CPQE in which the error occurred.		

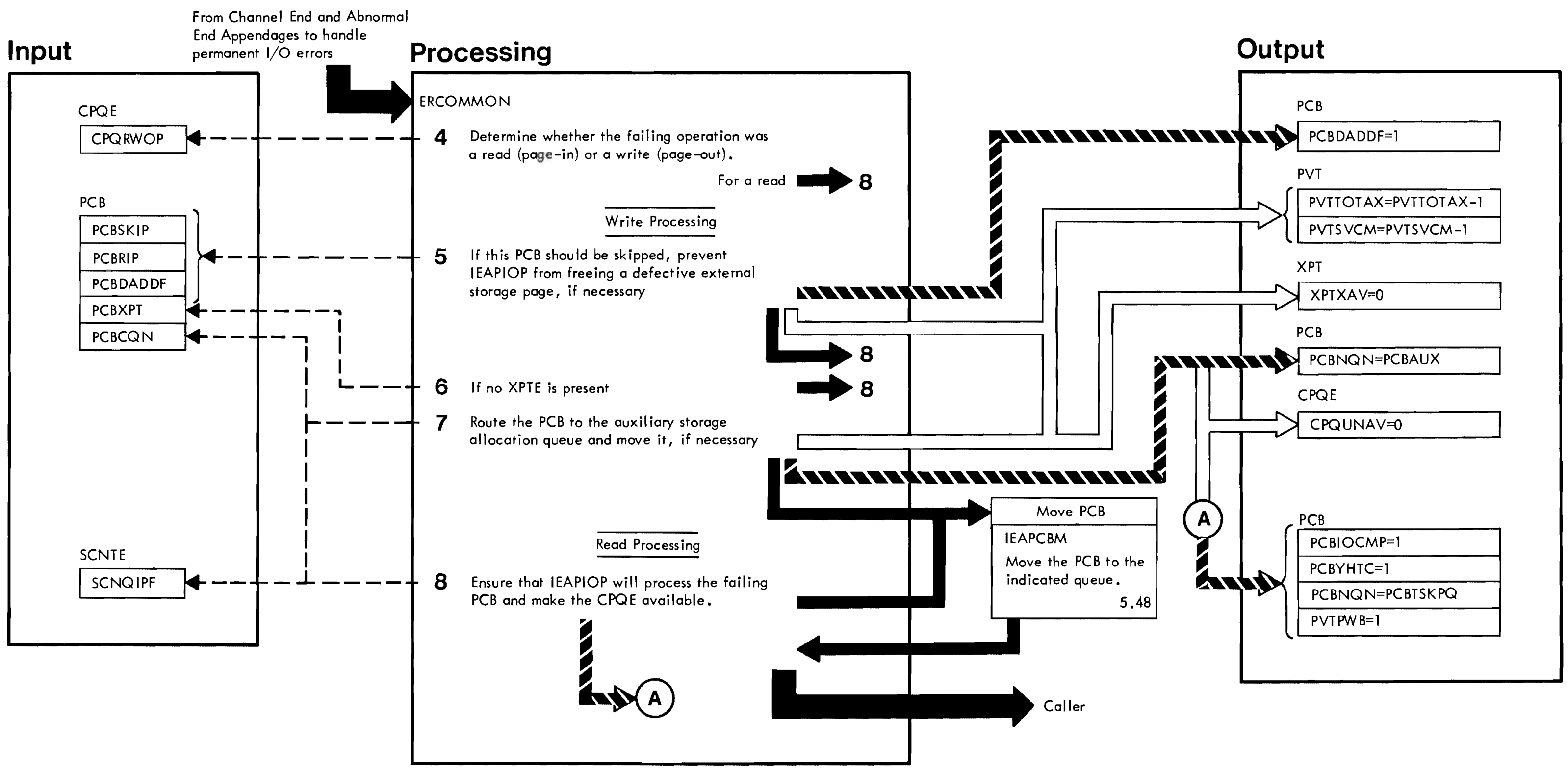
• **Diagram 5.56**
Paging Supervisor Appendages — Subroutines for Freeing Resources and Handling Errors



1 Determine whether this is the first CPQE on the chain. If not, handle the previous CPQE.

2 Locate the associated PCB, mark it complete, and route it.

3 Make this CPQE available.



4 Determine whether the failing operation was a read (page-in) or a write (page-out).

5 If this PCB should be skipped, prevent IEAPIOP from freeing a defective external storage page, if necessary

6 If no XPTE is present

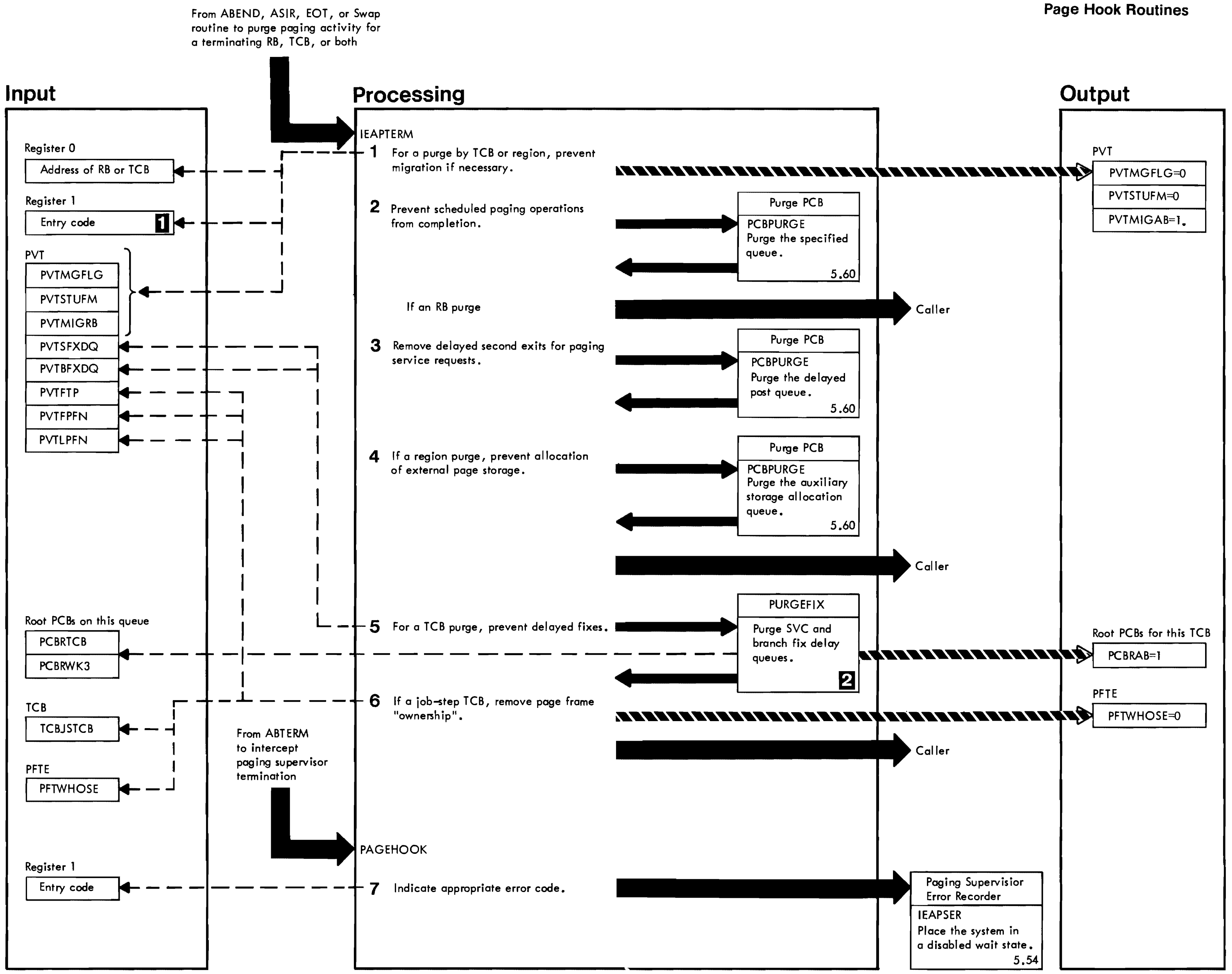
7 Route the PCB to the auxiliary storage allocation queue and move it, if necessary

8 Ensure that IEAPIOP will process the failing PCB and make the CPQE available.

Diagram 5.56 Paging Supervisor Appendages - Subroutines for Freeing Resources and Handling Errors (Mdoule IEAPPCIA)

NOTES	ROUTINE NAME	LABEL
1 When entered at this entry point, CBRET begins processing with the previous CPQE.	CBRET	MOR BAC
2 When entered at this entry point, CBRET begins processing with the current CPQE.		DOTHIS
If the PCB is on the I/O active queue (PCBCQN≠PCBNQN), the PCB is routed to the task post queue.		INDPOS
3 The post-work bit (PVTPWB) is turned on to trigger the posting mechanism for the completed PCBs.		
4	ERCOMMON	ERCOMMON
5 If PCBSKIP=1, PCBRIP=1, and PCBDADDF=1, IEAPIOP will free the external page storage associated with this PCB when the I/O is marked complete. Setting PCBDADDF=0 prevents this. The counts in the PVT are decremented to indicate one less page available for use in the page data set.		CERR00
6 PCBSKIP=0 and no XPTE exists is possible for swap-out pages.		
7 Routing the PCB to the auxiliary storage allocation queue causes another external storage page to be assigned to this PCB and retry to be attempted for the page-out operation.		CERR05
If the page PCB is on the I/O initiation queue, IEAPIOS will call Move PCB.		CERR02
8 If the PCB is on the Page I/O initiation queue, or if it is not and IEAPIOS is not processing (SCNQIPF=0), the PCB is routed to the task post queue only by setting PCBNQN. The I/O FLIH will call IEAPIOP when the post work bit (PVTPWB) is on. If IEAPIOS is processing, Move PCB is called to place the PCB on the task post queue.		CERR01 CERR03 CERR04 CERR05 CERR02

Diagram 5.57
Termination Interface and Page Hook Routines



1 For a purge by TCB or region, prevent migration if necessary.

2 Prevent scheduled paging operations from completion.

If an RB purge

3 Remove delayed second exits for paging service requests.

4 If a region purge, prevent allocation of external page storage.

5 For a TCB purge, prevent delayed fixes.

6 If a job-step TCB, remove page frame "ownership".

7 Indicate appropriate error code.

Diagram 5.57 Termination Interface and Page Hook Routines (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
1 If PVTMGFLG=1 and PVTSTUFM=1 and: • The purge is by TCB, they are set to 0. • The purge is by region, no action is taken. If PVTMGFLG=1 and PVTSTUFM=0, and the input TCB matches the TCB to be migrated (PVTMIGRB=Register 0), the ABEND migrate flag is set (PVTMIGAB=1).	IEAPTERM	PURGETCB
2 PURGEPCB is called once for each of the 3 PCB queues scanned (I/O active queue, real storage allocation queue, and I/O initiation queue).		PURGERB
5 PURGEFIX is called once for each queue purged.		
6 If TCBJSTCB=register 0, this is a job-step TCB. PFTWHOSE is set to 0 in every PFTE in which PFTWHOSE equals register 0.		TEST1
7 If register 1 = 0, the paging task was scheduled for termination for an external reason. Error code 2101 is passed to IEAPSER. 2100 is passed if register 1 = 4 (a program check occurred in the paging task).	PAGEHOOK	
$\boxed{1}$ Entry codes are: 0 - purge by TCB. 4 - purge by RB. 8 - purge by region.	IEAPTERM	
$\boxed{2}$ PURGEFIX marks root PCBs, associated with the FIX requests made by the task being terminated, for "ABEND interception." If the root PCB is for this TCB (PCBRTCB=input TCB address), its PCBRAB flag is set to 1. The next root PCB on the queue is accessed (via PCBRWK3) and the test is repeated. When the end of the queue is reached (PCBRWK3=0), return is made to the caller.	PURGEFIX	

Diagram 5.58
FIX Quiesce and Purge Routines

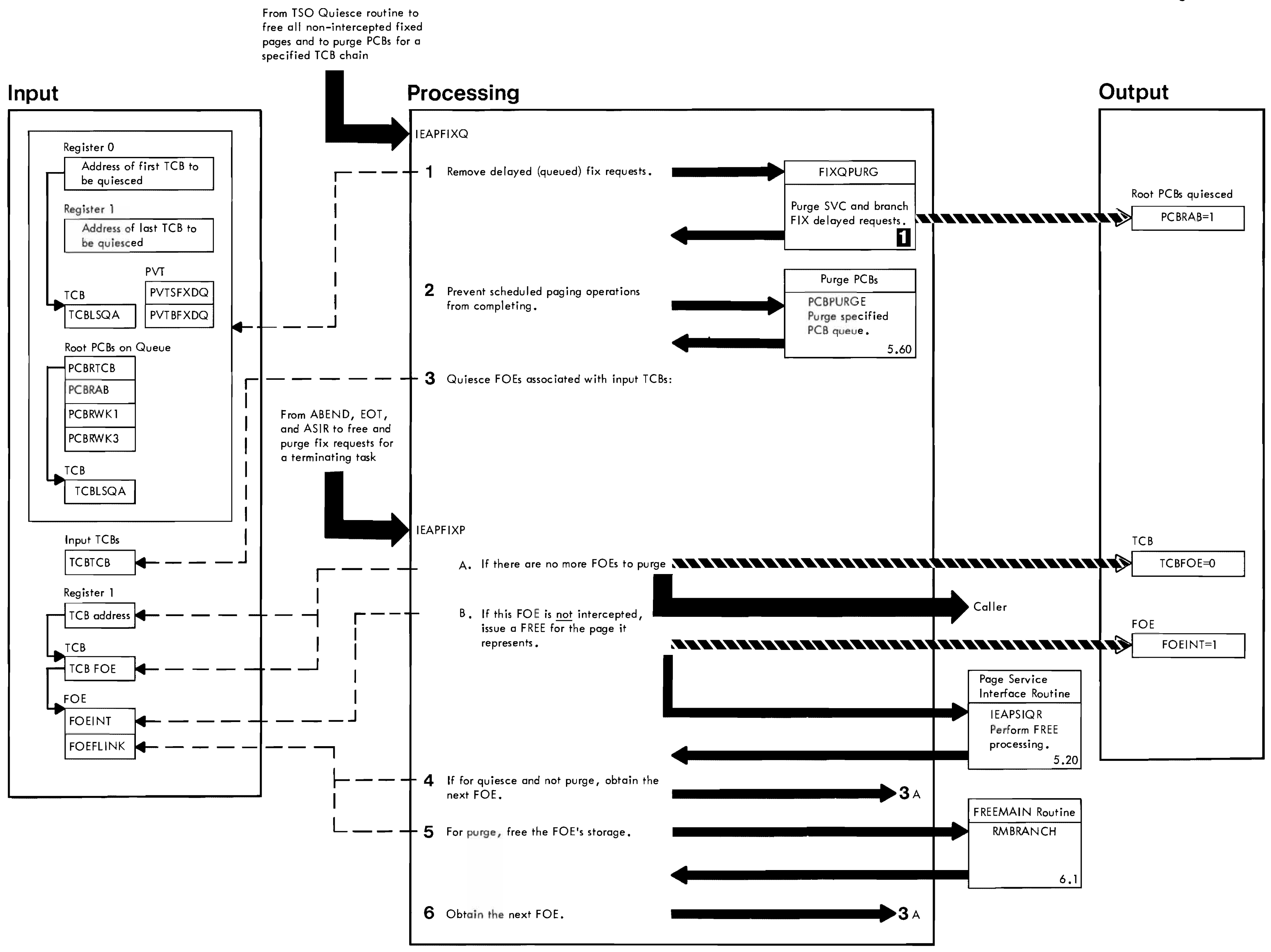
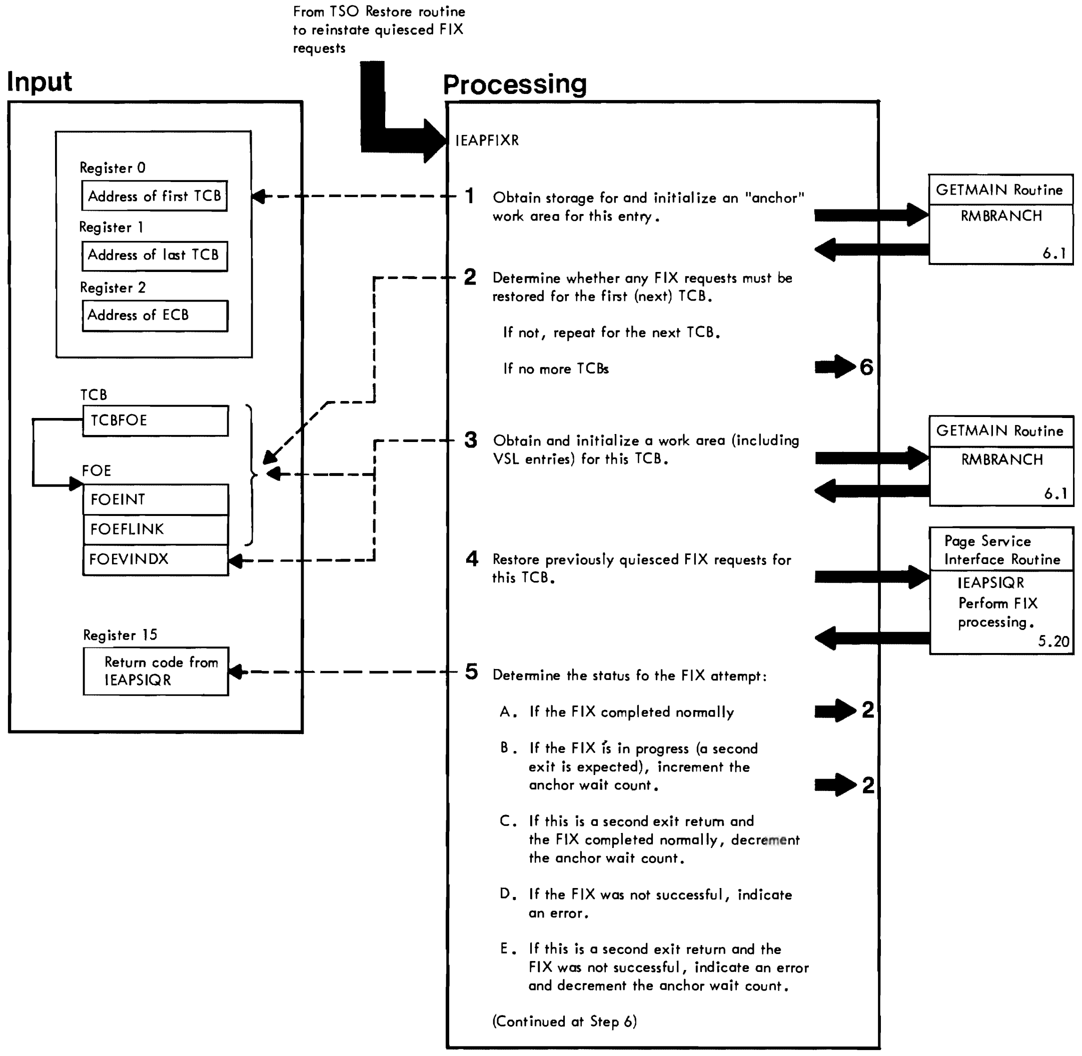

From TSO Quiesce routine to free all non-intercepted fixed pages and to purge PCBs for a specified TCB chain
Input
Processing
Output
Register 0
Address of first TCB to be quiesced
Register 1
Address of last TCB to be quiesced
PVT
PVTSFXDQ
PVTBFXDQ
TCB
TCBLSQA
Root PCBs on Queue
PCBRTCB
PCBRAB
PCBRWK1
PCBRWK3
TCB
TCBLSQA
Input TCBs
TCBTCB
Register 1
TCB address
TCB
TCB FOE
FOE
FOEINT
FOEFLINK
IEAPFIXQ
1 Remove delayed (queued) fix requests.
FIXQPURG
Purge SVC and branch FIX delayed requests.
1
2 Prevent scheduled paging operations from completing.
Purge PCBs
PCBPURGE Purge specified PCB queue.
5.60
3 Quiesce FOEs associated with input TCBs:
From ABEND, EOT, and ASIR to free and purge fix requests for a terminating task
IEAPFIXP
A. If there are no more FOEs to purge
Caller
B. If this FOE is not intercepted, issue a FREE for the page it represents.
Page Service Interface Routine
IEAPSIQR Perform FREE processing.
5.20
4 If for quiesce and not purge, obtain the next FOE.
3 A
5 For purge, free the FOE's storage.
FREEMAIN Routine
RMBRANCH
6.1
6 Obtain the next FOE.
3 A
Root PCBs quiesced
PCBRAB=1
TCB
TCBFOE=0
FOE
FOEINT=1

Diagram 5.58 FIX Quiesce and Purge Routines (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
1 FIXQPURG is called once for each queue purged.	IEAPFIXQ	
2 PURGEPCB is called once for each of the four PCB queues purged (page I/O active queue, I/O initiation queue, real storage allocation queue, and delayed post PCB queue). Quiesce is indicated for all the calls.		
3 This loop (FOEPURGE internal subroutine) is completed once for each input TCB.	IEAPFIXP FOEPURGE	CALLFOE
3A If entry is at IEAPFIXP, registers are restored before return. If entry is for quiesce (at FOEPURGE) return is immediate.		FOEDONE
3B If FOEINT=1, the FOE is either purged or quiesced already and no FREE is issued for it. Parameters passed to FREE (IEAPSIQR) are: Register 0 - TCB address; Register 1 - first half of VSL entry; Register 2 - second half of VSL entry		TESTINIT
5 Parameters passed to FREEMAIN are: Register 0 - X'FF000008'; Register 1 - FOE address; Register 3 - CVT address; Register 4 - TCB address; Register 5 - 0		PURGE
1 FIXQPURG quiesces activity on one of the FIX delay queues. TCBLSQA of the TCB pointed to by the first (next) root PCB (via PCBRTCB) is compared to TCBLSQA of the input TCB. If they are not equal, or if they are equal but the root PCB is marked for termination interception (PCBRAB=1), the next root PCB is accessed (via PCBRWK3), and the comparison is repeated. When a match is found, if the SVC FIX delay queue is being quiesced, the issuer of the FIX is posted. POST is called with the following parameters: Register 10 - post code = X'30'; Register 11 - ECB address (from PCBRWK1); Register 12 - TCB address (from PCBRTCB). Upon return, or if POST was not called (that is, if the branch FIX delay queue is being quiesced), PCBRAB is set to 1 and the search continues with the next root PCB.	FIXQPURG	



IEAPFIXR

1 Obtain storage for and initialize an "anchor" work area for this entry.

2 Determine whether any FIX requests must be restored for the first (next) TCB.

If not, repeat for the next TCB.

If no more TCBs

3 Obtain and initialize a work area (including VSL entries) for this TCB.

4 Restore previously quiesced FIX requests for this TCB.

5 Determine the status fo the FIX attempt:

A. If the FIX completed normally

B. If the FIX is in progress (a second exit is expected), increment the anchor wait count.

C. If this is a second exit return and the FIX completed normally, decrement the anchor wait count.

D. If the FIX was not successful, indicate an error.

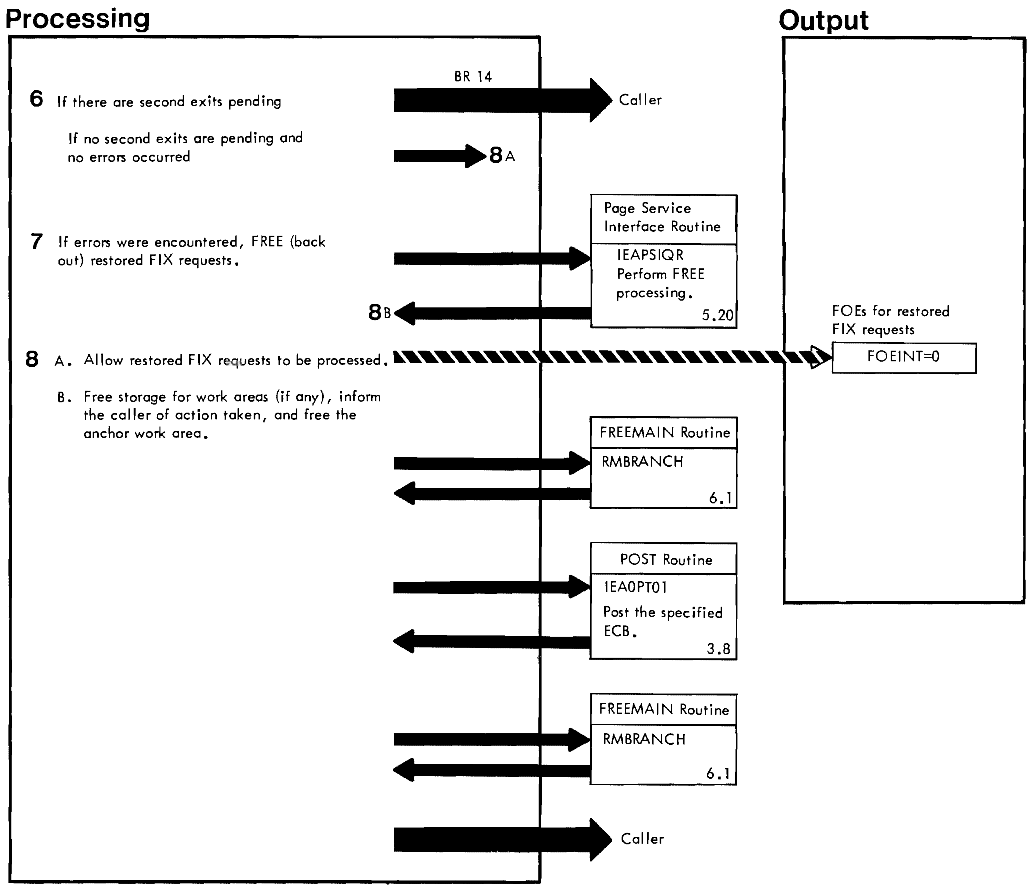
E. If this is a second exit return and the FIX was not successful, indicate an error and decrement the anchor wait count.

(Continued at Step 6)

Diagram 5.59 (Steps 1-5) FIX Restore Routine (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
1 The "anchor" contains information pertinent to this request. The work areas (including VSLs) for each TCB are chained to the anchor. Initially, registers are saved in it and the error flags, wait count (count of pending second exits), and post code fields are zeroed.	IEAPFIXR	SETNEG
2 Each FOE chained to this TCB is searched. An internal FOECOUNT is set to 0 and incremented for each intercepted FOE (FOE with FOEINT=1) on the chain. If at the end of the search FOECOUNT=0, the next TCB is accessed.		STARTFOE
3 The work area is obtained from subpool 255 and its length is (8*FOECOUNT)+8. In it, the TCB address is saved, the return code field is set to X'FF', and all flags are zeroed.		BUILDVSL
For each intercepted FOE on this chain, a VSL entry is built in the work area as follows: VSLSTART address is set to FOEVINDX. VSLEND1 is set to VSLSTART + 1. The flag byte is zeroed.		FOEMORE
4 The parameters passed to IEAPSIQR are: Register 0 - TCB address. Register 1 - Bit 0 - 1 (to indicate list entry) Bytes1-3 - First VSL entry address.		GOTOFIX
5 The return code from IEAPSIQR is placed in the work area and the anchor post code is set as follows: If the return code / The post code is <u>from IEAPSIOR is</u> / <u>set to</u> 0,32 — X'00' 4,28 — X'04' 8 — none 12,16,20,24,40 — X'0C' 36,44 — X'10' (For an explanation of the FIX return codes, see Diagram 5.21, Note 19.) A "no-FREE" flag in the work area, and the error flag in the anchor are set if any return code other than 0, 8, or 32 is received.		BTABLE

Diagram 5.59 (Steps 6-8)
FIX Restore Routine



6 If there are second exits pending

If no second exits are pending and no errors occurred

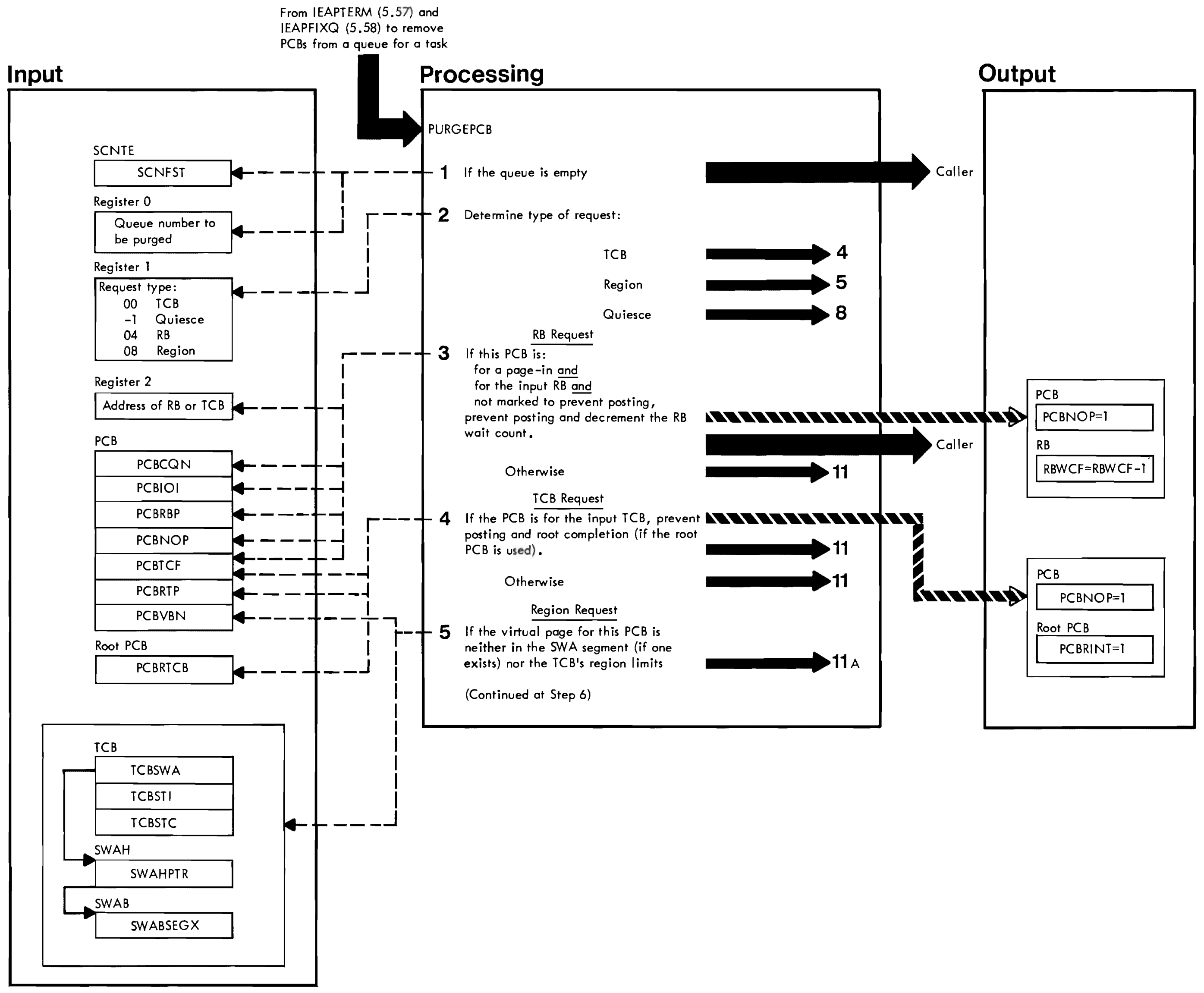
7 If errors were encountered, FREE (back out) restored FIX requests.

8 A. Allow restored FIX requests to be processed.

B. Free storage for work areas (if any), inform the caller of action taken, and free the anchor work area.

Diagram 5.59 (Steps 6-8) FIX Restore Routine (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
6 If the anchor wait count ≠ 0, second exits are pending.	IEAPFIXR	WCEQ0
7 The "no-FREE" flag is set in the work areas where errors were encountered because FIX backs out its own failing requests. The FIX Restore routine must back out (issue a FREE) for requests that completed successfully.		BACKMORE
IEAPSIQR is called once for each completed restore. The parameters passed to it are:		
VSL byte 0 - X'02' Register 0 - TCB address Register 1 - bit 0 - 1 bytes 1-3 - first VSL entry address.		
8 FREEMAIN is called once for each work area. If FIX requests are being backed out, FREEMAIN is called after each return from IEAPSIQR and then again for each work area not backed out (after the last return from IEAPSIQR).		FREEWA
Parameters passed to POST are:		POSTIT
Register 10 - post code Register 11 - ECB address Register 12 - TCB address.		



1 If the queue is empty

2 Determine type of request:

TCB

Region

Quiesce

RB Request

3 If this PCB is:
for a page-in and
for the input RB and
not marked to prevent posting,
prevent posting and decrement the RB wait count.

Otherwise

TCB Request

4 If the PCB is for the input TCB, prevent posting and root completion (if the root PCB is used).

Otherwise

Region Request

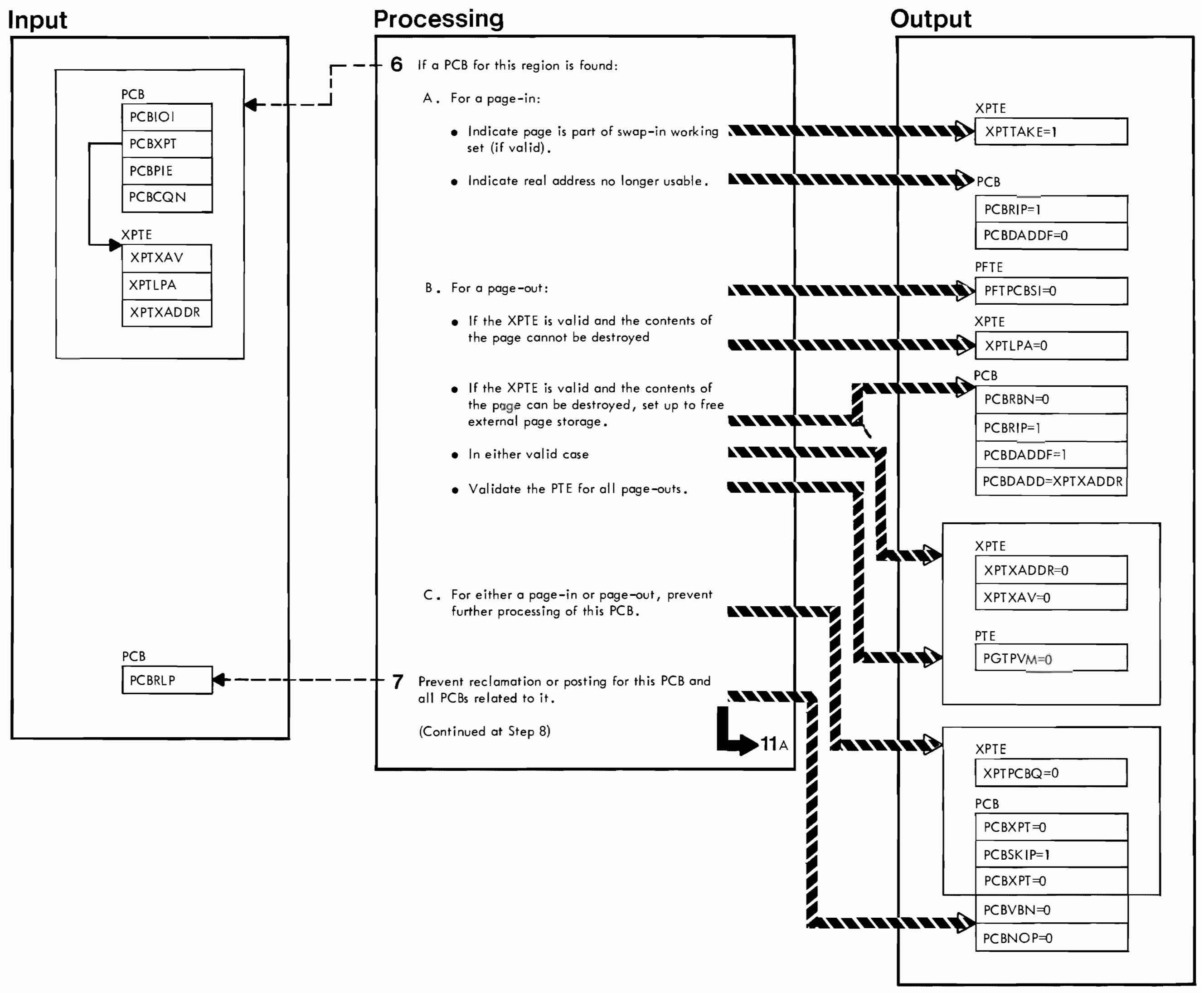
5 If the virtual page for this PCB is neither in the SWA segment (if one exists) nor the TCB's region limits

(Continued at Step 6)

Diagram 5.60 (Steps 1-5) Subroutine for Purging PCBs (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
1	PURGEPCB	
2		TESTTYPE
3 Since an RB can have only one outstanding paging request, the search is stopped as soon as it is found. The PCB is for a page-in if PCBIOI=1 or if the PCB is on the real storage allocation queue.		RBPURGE
4 If PCBTCF=0, the PCBRTP filed points to a root PCB and the TCB is found via PCBRTCB. If this route is taken, PCBRINT is set to 1.		TESTTCB
5 If SWAHPTR=0, no SWA segment exists. The SWA limit is in SWABSEGX. The TCB limit begins with the first segment number in TCBSTI and continues for TCBSTC segments.		REGION

Diagram 5.60 (Steps 6-7)
Subroutine for Purging PCBs



6 If a PCB for this region is found:

A. For a page-in:

- Indicate page is part of swap-in working set (if valid).
- Indicate real address no longer usable.

B. For a page-out:

- If the XPTE is valid and the contents of the page cannot be destroyed
- If the XPTE is valid and the contents of the page can be destroyed, set up to free external page storage.
- In either valid case
- Validate the PTE for all page-outs.

C. For either a page-in or page-out, prevent further processing of this PCB.

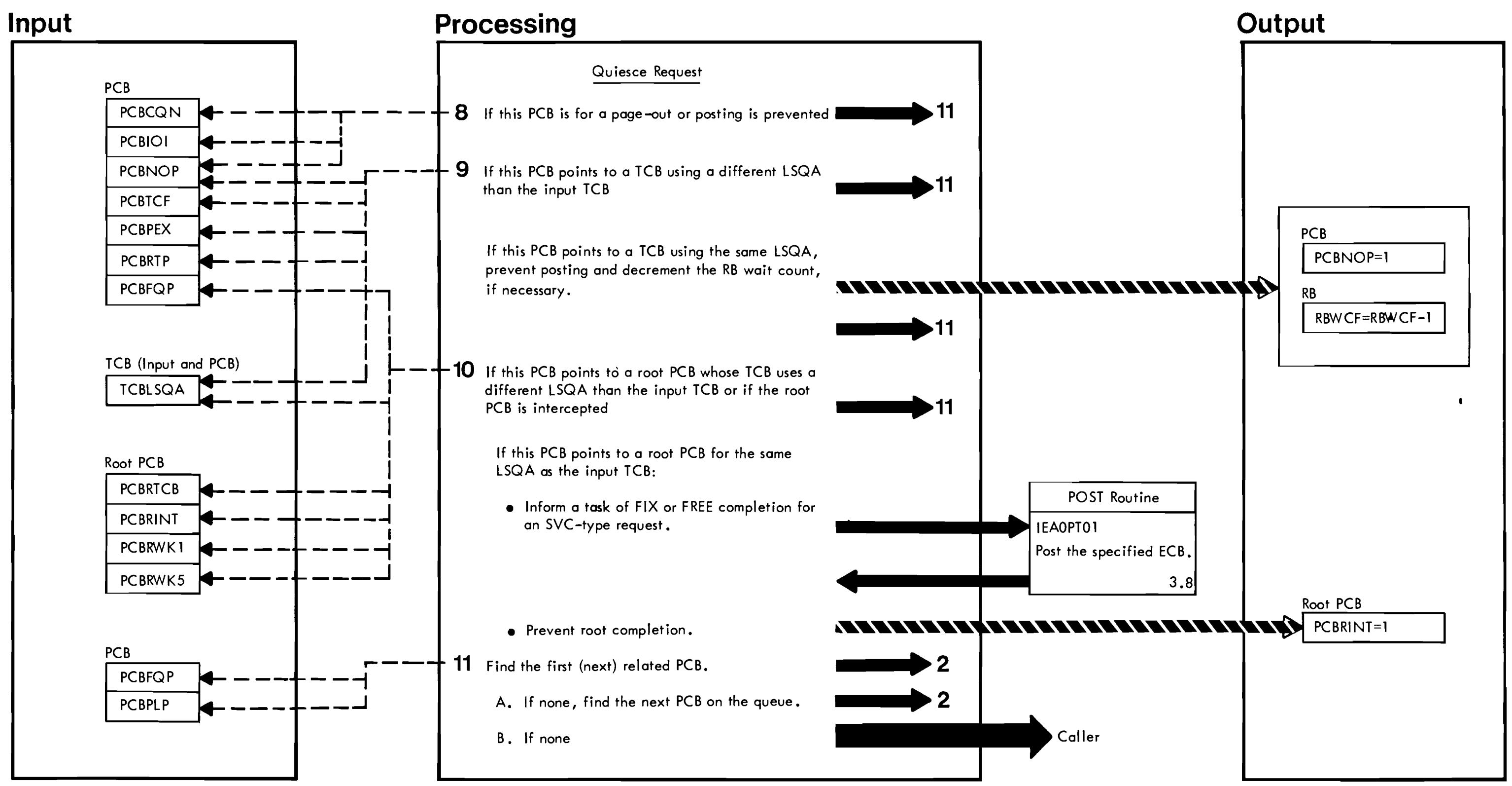
7 Prevent reclamation or posting for this PCB and all PCBs related to it.

(Continued at Step 8)

Diagram 5.60 (Steps 6-7) Subroutine for Purging PCBs (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
6 The PCB is for a page-in if PCBIOI=1 or if the PCB is on the real storage allocaticn queue.	PURGEPCB	INOUTTS
7		PCBVBNO

Diagram 5.60 (Steps 8-11)
Subroutine for Purging PCBs



Quiesce Request

8 If this PCB is for a page-out or posting is prevented

9 If this PCB points to a TCB using a different LSQA than the input TCB

If this PCB points to a TCB using the same LSQA, prevent posting and decrement the RB wait count, if necessary.

10 If this PCB points to a root PCB whose TCB uses a different LSQA than the input TCB or if the root PCB is intercepted

If this PCB points to a root PCB for the same LSQA as the input TCB:

- Inform a task of FIX or FREE completion for an SVC-type request.

- Prevent root completion.

11 Find the first (next) related PCB.

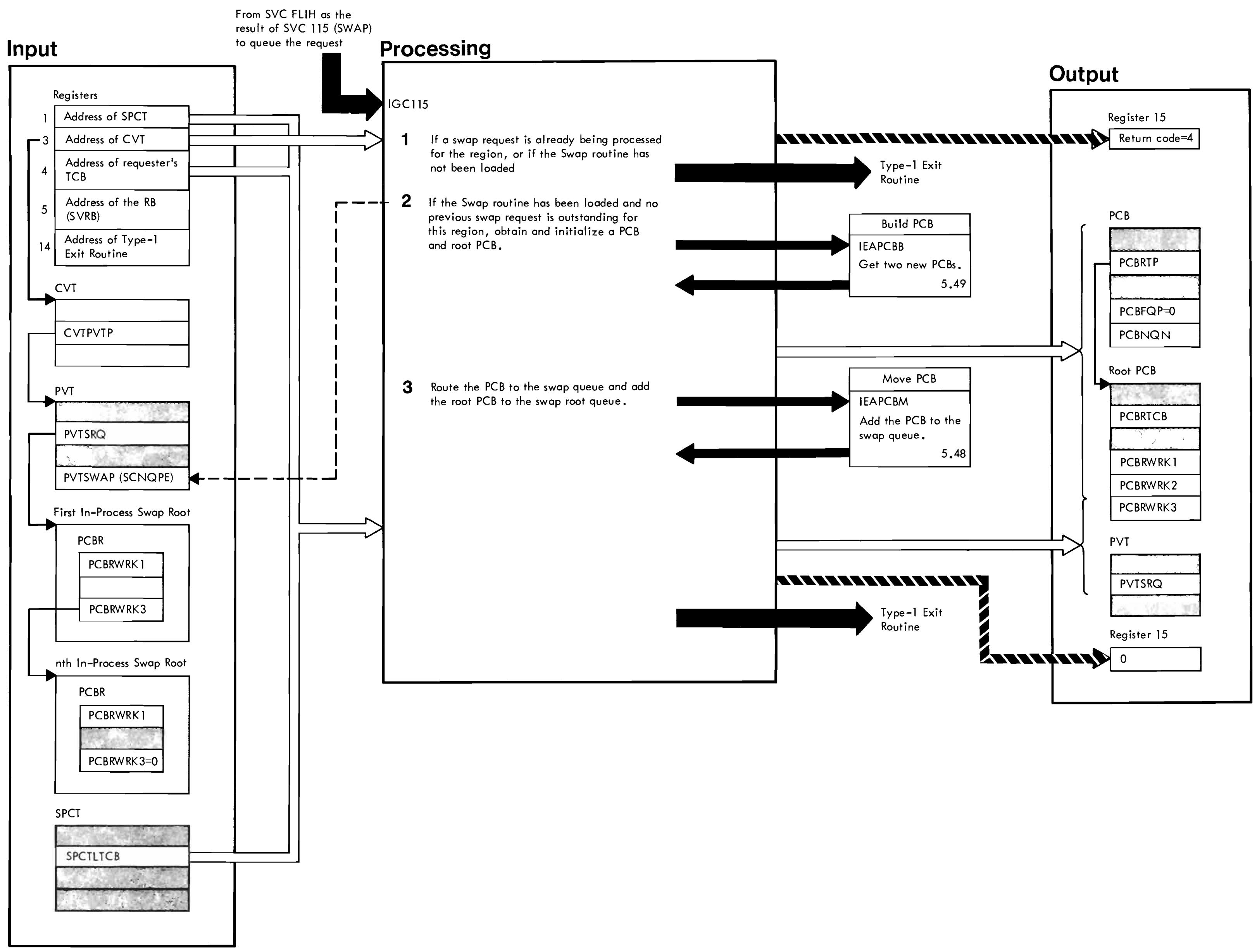
A. If none, find the next PCB on the queue.

B. If none

Diagram 5.60 (Steps 8-11) Subroutine for Purging PCBs (Module IEAPTERM)

NOTES	ROUTINE NAME	LABEL
8 The PCB is for a page-in if PCBICI=1 or if the PCB is on the real storage allocation queue.	PURGEPCB	QUI
9 The TCBLSQA fields of the input TCB and the TCB indicated in the PCB (by PCBRTP) are compared. If they are equal, this PCB must be quiesced. If the PCB to be quiesced is for a missing page exception (PCBPEX=1) and is not marked to prevent posting (PCBNOP=0), PCBNOP is turned on to prevent posting and the RB wait count (RBWCF) is decremented.		QUIIN
10 The TCBLSQA fields of the input TCB and the TCB indicated by the root PCB (via PCBRTCB) are compared as above. If they are equal and PCBRWK5 of the root PCB is 0, this is an SVC root and must be posted. The parameters passed to POST are: Register 10 - post code = 0 Register 11 - ECB address (from PCBRWK1) Register 12 - TCB address (from PCBRTCB).		QUITCTST
11		RELATED

Diagram 5.61
Swap SVC Interface Routine



1 If a swap request is already being processed for the region, or if the Swap routine has not been loaded

2 If the Swap routine has been loaded and no previous swap request is outstanding for this region, obtain and initialize a PCB and root PCB.

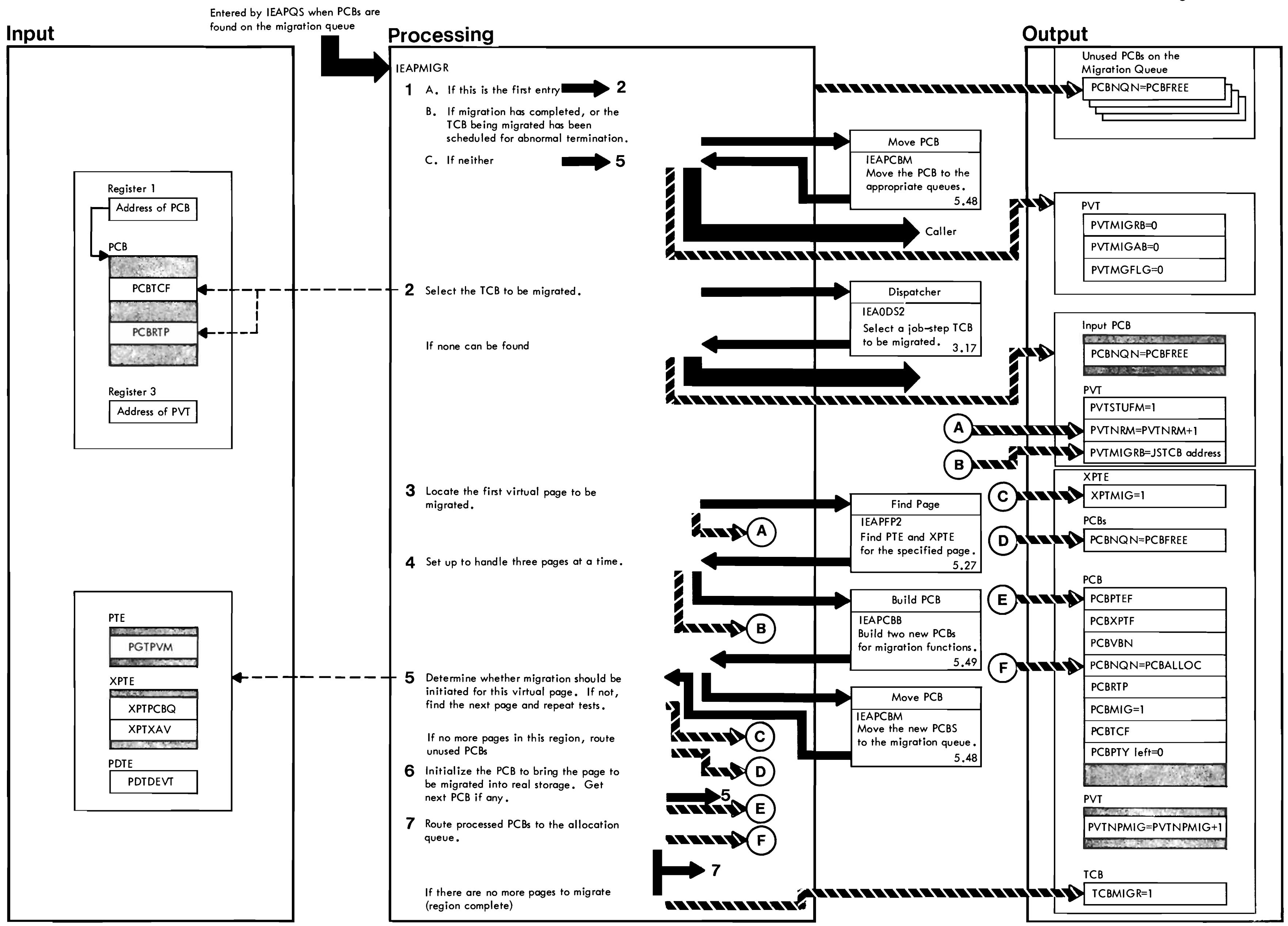
3 Route the PCB to the swap queue and add the root PCB to the swap root queue.

Diagram 5.61 Swap SVC Interface Routine (Module IEAPSSVC)

NOTES	ROUTINE NAME	LABEL
1 The address of the TCB from the input SPCT (SPCTLTCB) is compared to the TCB address in PCBRWK1 of each root PCB on the swap root queue (pointed to by PVTSRQ and chained by the PCBRWK3 fields of each root). If a match is found, the Swap routine is already in progress for this region.	IEAPSSVC	TESTTCB
If SCNQPE in the swap queue header (PVTSWAP) equals 0, the Swap routine has not been loaded.		LOADCHK

2 The PCB is routed to the swap queue (PCBNQN=PCBSWAP). The root PCB for this swap request is set up as follows:
PCBRTCB=input TCB address (from Register 4)
PCBRWK1=SPCTLTCB
PCBRWK2=SPCT address (from Register 1)
PCBRWK3=PVTSRQ.		

3 The new swap root PCB is added to the head of the swap root queue.		



IEAPMIGR

1 A. If this is the first entry → 2

B. If migration has completed, or the TCB being migrated has been scheduled for abnormal termination.

C. If neither → 5

2 Select the TCB to be migrated.

If none can be found

3 Locate the first virtual page to be migrated.

4 Set up to handle three pages at a time.

5 Determine whether migration should be initiated for this virtual page. If not, find the next page and repeat tests.

If no more pages in this region, route unused PCBs

6 Initialize the PCB to bring the page to be migrated into real storage. Get next PCB if any.

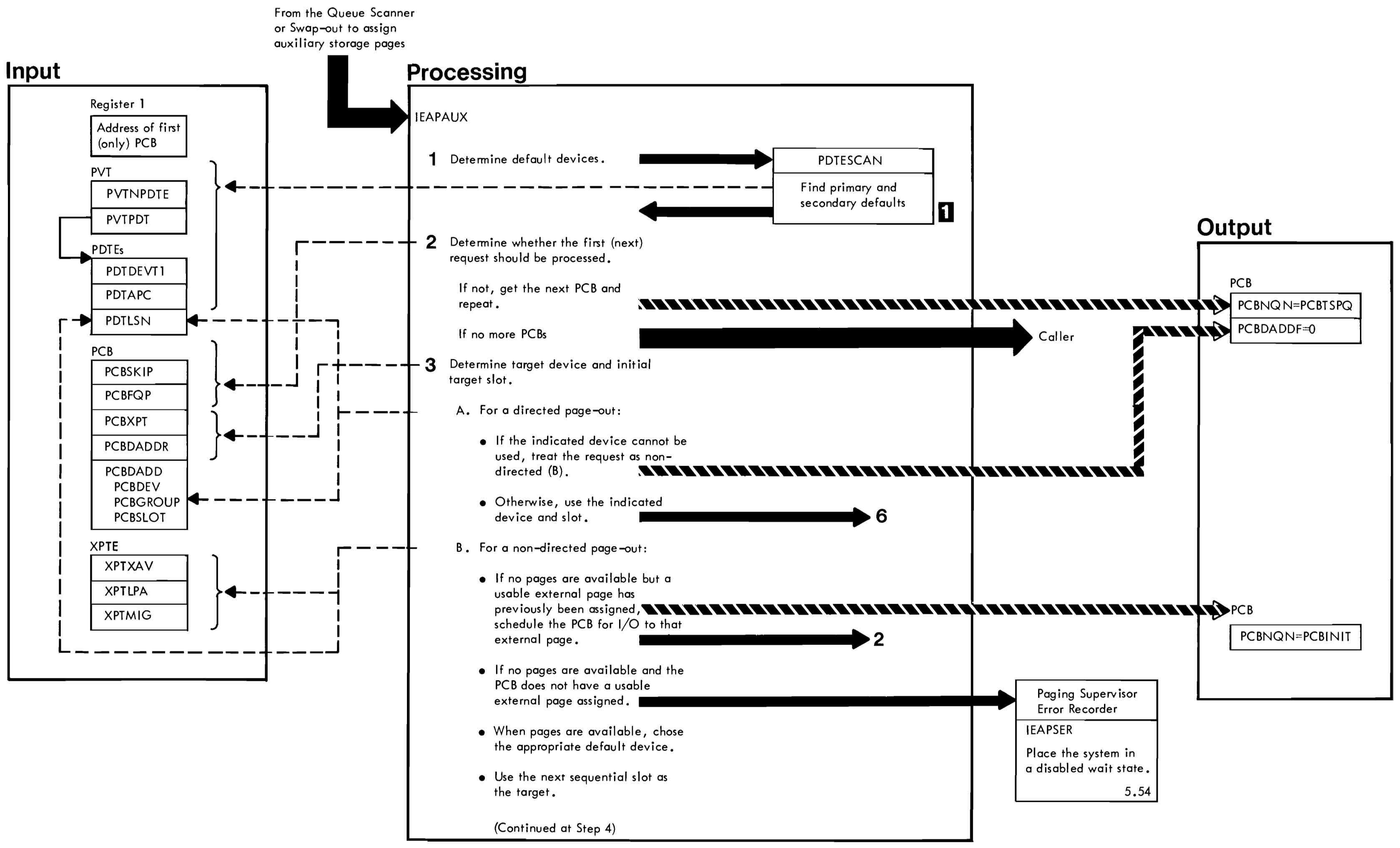
7 Route processed PCBs to the allocation queue.

If there are no more pages to migrate (region complete)

Diagram 5.62 Migration Routine (Module IEAPMIGR)

NOTES	ROUTINE NAME	LABEL
1 If this is not the first entry, and either migration has been finished or ABEND is indicated, all PCBs remaining on the migration queue are routed to the free queue, internal flags are reinitialized, and control is returned to the Queue Scanner.	IEAPMIGR	SNDENTRY
2 If the Swap routine has selected a TSO task to migrate, the TCB address in the PCB is used and the dispatcher is not called.		SETCBPTR
If the dispatcher could not select a JSTCB (return code=4), PVTSTUFM is set to indicate that the Swap routine must select a TCB, the input PCB is routed to the free queue, and control is returned to the Queue Scanner.		
3 The Find page routine is passed the address of the first virtual page in the region to be migrated and returns the PTE address in register 0 and the XPTE address in register 1.		SETSAVE
4 The second entry flag is set (checked in Step 1) to prevent building more PCBs when the Migration routine is reentered.		
5 Migration is not initiated if this page is in real storage, if paging is in progress for this page, if there is no external page storage present for this page, or if the external page storage is already on a secondary paging device.		MARKMIGR
6 This migration PCB has the lowest paging pricrity.		
7 PCBs representing pages to be migrated are moved to the auxiliary storage allocation queue.		

Diagram 5.63 (Steps 1-3)
Auxiliary Storage Manager
(Handling External Page Storage)



IEAPAUX

1 Determine default devices.

2 Determine whether the first (next) request should be processed.

If not, get the next PCB and repeat.

If no more PCBs

3 Determine target device and initial target slot.

A. For a directed page-out:

- If the indicated device cannot be used, treat the request as non-directed (B).
- Otherwise, use the indicated device and slot.

B. For a non-directed page-out:

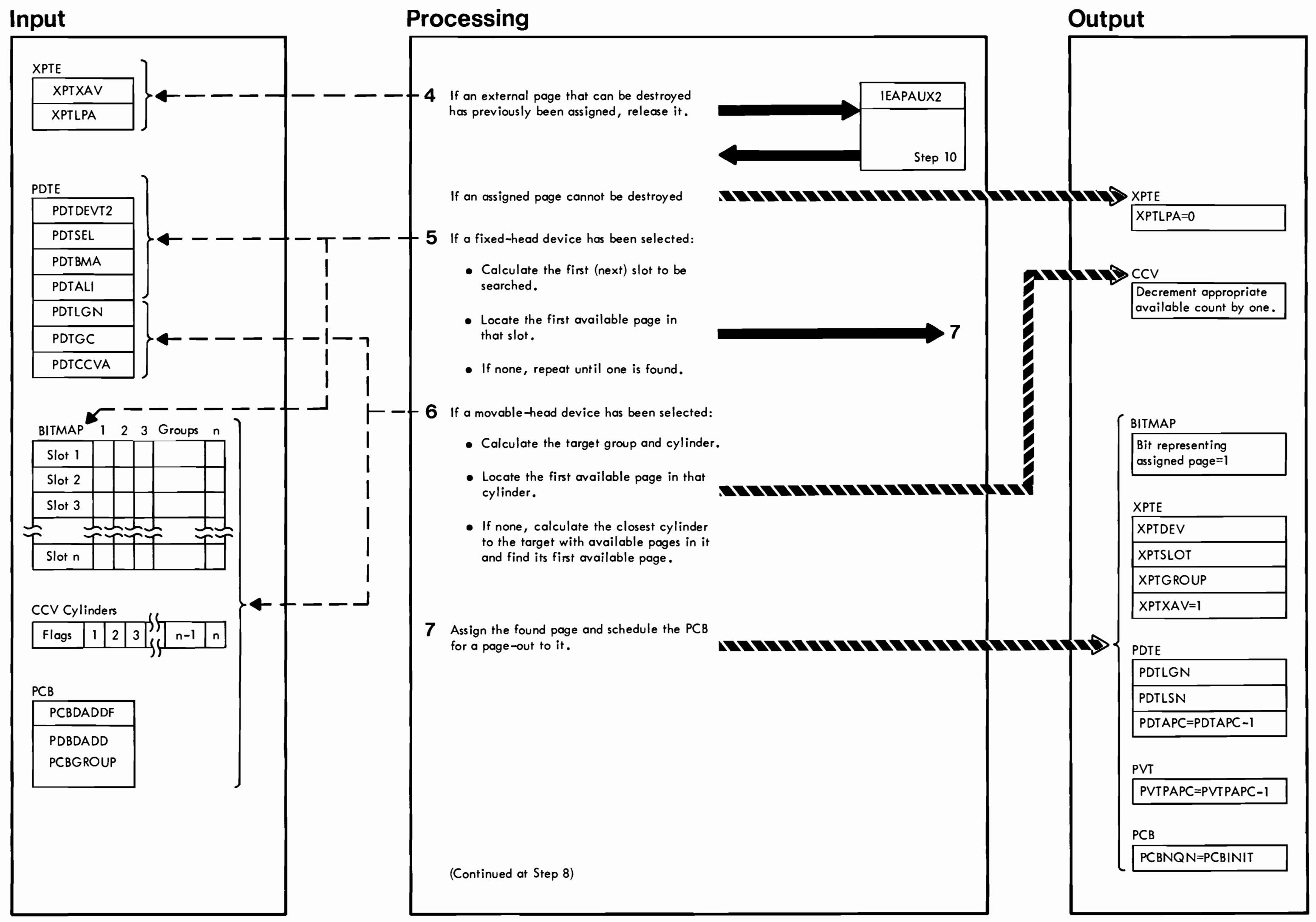
- If no pages are available but a usable external page has previously been assigned, schedule the PCB for I/O to that external page.
- If no pages are available and the PCB does not have a usable external page assigned.
- When pages are available, chose the appropriate default device.
- Use the next sequential slot as the target.

(Continued at Step 4)

Diagram 5.63 (Steps1-3) Auxiliary Storage Manager (Module IEAPAUXS)

NOTES	ROUTINE NAME	LABEL
1 The primary and secondary paging devices with the highest available page counts are chosen.	IEAPAUXS	IEAPAUXS
2 If PCBSKIP=1, the PCB is routed to the task post queue and the next PCB is accessed through PCBFQP.		PROCESS
3 For a directed page-out (PCBDADDF=1), if the indicated group number is not valid or if the device has no available pages, the indicated device cannot be used. Ⓐ If the device can be used, PCBDEV and PCBSLOT point to the targets.		XPTEBASE
For a non directed page-out, Ⓑ if both defaults are 0 but an external page was previously assigned to the virtual page (XPTXAV=1), the page-out is scheduled to the previously assigned external page. If, a page was not previously assigned, or if the page assigned cannot be destroyed (XPTLPA=1), IEAPSER is called with an error code 1500 and a major error indication. No page-out is initiated.		TESTDEFS
When pages are available (defaults≠0), if XPTMIG=1, the secondary default is used if one was designated. In all other cases, the primary default is used as the target.		AVAILPGS
The target slot is the next sequential slot after the last one used on the default device (PDTLSN+1).		CALLSLOT
1 PDTESCAN searches every PDTE comparing the PDTAPC fields of primary devices and secondary devices. The requested defaults are passed back to IEAPAUXS. A default device is a paging device from which a page slot is to be assigned for a page-out operation when a particular device has not been specified in the page-out request. Ⓐ A directed page-out is a page-out operation in which the page slot is to be assigned from a paging device specified in the page-out request (rather than from a default device). Ⓑ A page-out operation in which the page slot is to be assigned from a default device, rather than from a particular device specified in the page-out request. If there are no available pages (either primary or secondary), that default is designated as 0.	PDTESCAN	

**Diagram 5.63 (Steps 4-7)
Auxiliary Storage Manager
(Handling External Page Storage)**



4 If an external page that can be destroyed has previously been assigned, release it.

If an assigned page cannot be destroyed

5 If a fixed-head device has been selected:

- Calculate the first (next) slot to be searched.
- Locate the first available page in that slot.
- If none, repeat until one is found.

6 If a movable-head device has been selected:

- Calculate the target group and cylinder.
- Locate the first available page in that cylinder.
- If none, calculate the closest cylinder to the target with available pages in it and find its first available page.

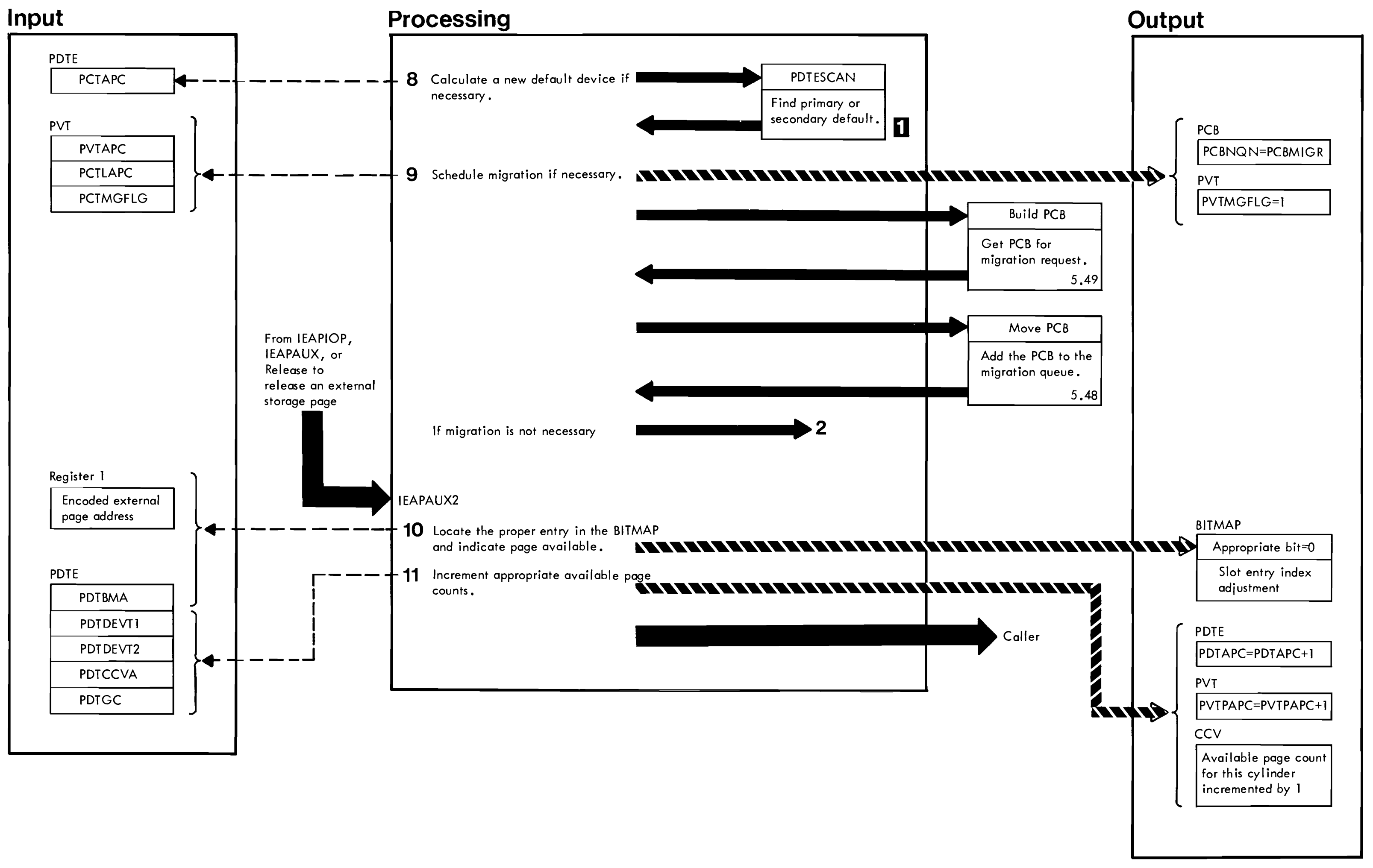
7 Assign the found page and schedule the PCB for a page-out to it.

(Continued at Step 8)

Diagram 5.63 (Steps 4-7) Auxiliary Storage Manager (Module IEAPAUXS)

NOTES	ROUTINE NAME	LABEL
4 If XPTXAV=1 and XPTLPA=0, the external page is released.	IEAPAUXS	FREEAUXS
5 For a fixed head device, an unrestricted search is performed using all slots on a device if necessary: 1 The highest slot index value (the slot index is the first byte of the slot in the BITMAP that may contain an available page) is calculated (PDTSEL-2). 2 The target slot index address = PDTBMA+ (target slot number-1)*PDTSEL. 3 The address of the byte to be examined first= the target slot index address+index value+1. 4 If the current index value e the high index value (from above), the next slot to be searched is accessed using PDTALI as the increment. 5 When a slot index containing less than the high value is found, the byte it indicates is examined for a bit with a zero value. This bit indicates the group (external page) to be used.		TESTTYPE
6 For a movable-head device, a restricted search is per- formed using only the part of each slot in the indicated cylinder: 1 For a directed page-out, PCBGROUP indicates the target group. For a nondirected page-out, the target group is the last one assigned on the default device (PDTLGN). 2 The target cylinder=((PDTLGN-1)/PDTGC)+1. 3 The CCV is accessed via PVTCCVA. If the byte rep- resenting this cylinder = 0, there are no available pages in the target. 4 The closest cylinder to the target with available pages is found by testing the CCV byte for the next higher cylinder, then the next lower, then 2 higher, 2 lower, etc. 5 When a cylinder with available pages in it (either the original target or one close to it) is found, the low group number (first bit in the byte to be searched) is calculated. The bits in the BITMAP that represent the target cylinder are then searched for a 0 bit. If none, the slot to be searched is incre- mented by PDTALI (at the end of each cylinder rather than at the end of the slot as in the unrestricted search). The bits in the next slot entry for this cylinder is then searched. This continues until an available page is found.		MOVEHEAD
7 The bit in the BITMAP representing the found page is turned on. The device, slot, and group numbers (the encoded address) are placed in the XPTE and the page assigned flag (XPTXAV) is turned on. The last used slot and group numbers (PDTLSN and PDTLGN respectively) are incremented appropriately. The slot index is also adjusted as necessary (for fixed-head devices only). The available page count for the device (PDTAPC) is decremented and, if the device is a primary device, the primary available page count (PVTPAPC) is also decre- mented. The PCB is then routed to the I/O Initiation Queue.		FOUND

**Diagram 5.63 (Steps 8-11)
Auxiliary Storage Manager
(Handling External Page Storage)**


Input
PDTE
PCTAPC
PVT
PVTAPC
PCTLAPC
PCTMGFLG
Register 1
Encoded external page address
PDTE
PDTBMA
PDTDEVT1
PDTDEVT2
PDTCCVA
PDTGC
Processing
From IEAPIOP, IEAPAUX, or Release to release an external storage page
IEAPAUX2
PDTESCAN
Find primary or secondary default.
1
Build PCB
Get PCB for migration request.
5.49
Move PCB
Add the PCB to the migration queue.
5.48
2
Caller
Output
PCB
PCBNQN=PCBMIGR
PVT
PVTMGFLG=1
BITMAP
Appropriate bit=0
Slot entry index adjustment
PDTE
PDTAPC=PDTAPC+1
PVT
PVTPAPC=PVTPAPC+1
CCV
Available page count for this cylinder incremented by 1

8 Calculate a new default device if necessary.

9 Schedule migration if necessary.

If migration is not necessary

10 Locate the proper entry in the BITMAP and indicate page available.

11 Increment appropriate available page counts.

Diagram 5.63 (Steps 8-11) Auxiliary Storage Manager (Module IEAPAUXS)

NOTES	ROUTINE NAME	LABEL
8 If the page assignment has caused the available page count for that device to go to 0, PDTESCAN is called to designate a new default.	IEAPAUXS	
9 If the new primary available page count (PVTPAPC) is below the low threshold (PVTLAPC) and migration is not already in progress (PVTMGFLG=0) and pages are available on a secondary paging device, migration is scheduled.		ENABLE
10 The PDTE address for the device containing the page to be released is calculated and the BITMAP is accessed (via PDTBMA). The appropriate bit is set to 0 and the slot entry index is adjusted if the index of the released page is lower than the current index value.	IEAPAUX2	IEAPAUX2
11 The counts are updated as follows: PDTAPC=PDTAPC+1. If the page was released from a primary paging device, PVTPAPC=PVTPAPC+1. If the page was released from a movable-head device, the cylinder count in the CCV is located (PDTCCVA+2+(group number-1)/PDTGC) and incremented.		UPDTAPC

SECTION 6

Virtual Storage Supervision

Virtual storage is the name given to the entire span of addresses available on a System/370 with the dynamic address translation feature enabled. The size of virtual storage is equal to the size of real storage when the system is operating in BC (basic control) mode or EC (extended control) mode with translation disabled. But when the system is in EC mode with translation enabled, the size of virtual storage is limited only by the addressing capability of the system, not by the size of real storage.

Like other system resources, virtual storage can be shared by many users. Consequently, the allocation of space within virtual storage must be supervised. Space must be allocated to a user when it is needed and freed when it is no longer needed. The supervisor routines that control the allocation and release of virtual storage are collectively referred to as the VSS (virtual storage supervision) routines.

The VSS routines service two macro instructions: GETMAIN (which is used to allocate storage) and FREEMAIN (which is used to release storage that was allocated previously). When executed, each macro instruction results in an SVC interruption and passage of control to the appropriate VSS routines. For an overview of the method of operation of the VSS routines, see Diagram 6.1.

Requests for allocation of virtual storage are serviced by a subset of the VSS routines called the GETMAIN routines. (There is no actual routine with the entry-point name GETMAIN.) These routines service all requests for virtual storage, including requests for a new region, space within an existing region, space within a local system queue area, space for a system work area, and space within the system queue area. The GETMAIN routines create, reference, and continually update queues of control blocks to determine whether a request for storage can be satisfied, and from where it can be satisfied. The GETMAIN routines pass the address of the allocated area to the requesting routine (in general register 1 if an SVC 10 instruction caused entry, or in a user-specified location if an SVC 4 instruction caused entry). For an overview of the method of operation for allocating storage, see Diagram 6.2.

Requests to free virtual storage are serviced by a subset of the VSS routines called FREEMAIN routines. These routines update control block queues to reflect the release of previously allocated space, thereby making the space available for reallocation. The FREEMAIN routines service all requests to free virtual storage, including requests to free an entire region, space within a region, space within a local system queue area, space within a system work area, and space within the system queue area. For an overview of the method of operation for releasing storage, see Diagram 6.27.

The VSS routines assign blocks of storage to the various tasks according to their needs. The virtual storage supervision routines:

- Allocate storage blocks on request.
- Release storage blocks on request.
- Ensure that sufficient external page storage exists to back up batch-processing and TSO jobs.
- Ensure that fixed page-frames exist for all SQA, LSQA, and nonpageable (V=R) region space allocated.
- Maintain a record of storage allocated from supervisor-owned storage, when requested.
- Monitor requests for user storage and storage in subpools 251 and 252 (user's job pack area).
- Maintain storage usage information.
- Protect storage with fetch protection and storage protection keys.

SUBPOOLS

A subpool is a group of logically related storage blocks identified by a subpool number. The subpool number indicates to the VSS routines the kind of storage that is requested. Figure 6-1 summarizes the subpool assignments.

Subpool Number	Indicates Request for	Attributes of Subpool	Notes
0-127	Space within a region	Job-oriented Job step's protection key Fetch-protected	These are the only valid subpool numbers for programs running in problem state. A request for a higher number will cause the problem program to be abnormally terminated. When requested by programs in supervisor state and key 0, subpool 252 is assigned.
128			Reserved for compatibility with VS1.
129-231			Undefined.
232	External page	For TSO use only	To reserve external page storage for a TSO task.
233	Space within LSQA (task-related)	Job-oriented Fixed Protection key=0 Task-related Swappable Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire accountable, fixed, protected storage that is job-oriented and freed at end of task.
234	Space within LSQA (job-step-related)	Job-oriented Fixed Protection key=0 Job-step-related Swappable Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire accountable, fixed, protected storage that is job-oriented and freed at end of job step.
235	Space within LSQA (explicitly assigned and freed)	Job-oriented Fixed Protection key=0 Explicitly assigned and freed Not fetch-protected Swappable	Allows a task running in key 0 or supervisor state to acquire nonaccountable, fixed, protected storage that is job-oriented.
236	Space for SWA	For system use only Protection key=0 Fetch-protected	To assign or free a segment of pageable virtual storage for the system work area.
237	Space within SWA	For system use only Protection key=0 Fetch-protected	To assign or free a SWA page.
238			Reserved for compatibility with VS1.
239			Undefined.

Figure 6-1 (Part 1 of 3). Table showing attributes and uses of subpools

Subpool Number	Indicates Request for	Attributes of Subpool	Notes
240	Space within a region (job-step-related)	Job-oriented Pageable User's protection key Fetch-protected Job-step-related	Treated as subpool 250 to maintain compatibility with MFT. Automatically freed at end of step.
241	Space within SQA (explicitly assigned and freed)	System-oriented Pageable Protection key=0 Explicitly assigned and freed Not fetch-protected	Mapped into subpool 245.
242	Nonpageable region	For scheduler use only	A new nonpageable region is assigned or an existing nonpageable region is freed.
243	Space within SQA (task-related)	System-oriented Fixed Protection key=0 Task-related Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire accountable, fixed, protected storage that is system-oriented and freed when the task terminates.
244	Space within SQA (job-step-related)	System-oriented Fixed Protection key=0 Job-step-related Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire accountable, fixed, protected storage that is system-oriented and freed when the job step terminates.
245	Space within SQA (explicitly assigned and freed)	System-oriented Fixed Protection key=0 Explicitly assigned and freed Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire nonaccountable, fixed, protected storage that is system-oriented.
246			Reserved. Used in MVT to exchange regions.
247	Pageable region	For scheduler use only	A new pageable region is assigned or an existing pageable region is freed. External page storage allocation is assumed when using this subpool.
248			Reserved. Used in MVT for rollout/rollin.
249	LSQA segments	For scheduler use only	New segments of virtual storage are assigned for LSQA, or existing segments are freed.

Figure 6-1 (Part 2 of 3). Table showing attributes and uses of subpools

Subpool Number	Indicates Request for	Attributes of Subpool	Notes
250	Space within a region	Job-oriented User protection key Fetch-protected	Allows a task running in supervisor state to acquire unprotected storage in the user's region. All subpool 250 requests are assigned subpool 0 (high end of the region) of the associated task.
251	Space within a region	Job-oriented User's protection key Job-step-related Protected	Allows a task running in key 0 or supervisor state to acquire accountable, unprotected, pageable storage in the low-address end of the user's partition. Space is job-oriented and automatically freed at the termination of the job step. Used for nonreenterable modules in the job pack area.
252	Space within a region	Job-oriented Protection key=0 Job-step-related Swappable Not fetch-protected	Allows a task running in key 0 and supervisor state to acquire accountable, pageable, protected storage in the high address end of the user's region that is job-oriented and automatically freed at the termination of the job-step task. Used for reenterable modules from the SYS1.SVCLIB or SYS1.LINKLIB.
253	Space within LSQA (task-related)	Job-oriented Fixed Protection key=0 Task-related Not fetch-protected Swappable	Allows a task running in key 0 or supervisor state to acquire fixed, accountable, protected storage in the LSQA for the user's region that is job-oriented and freed when the task terminates.
254	Space within LSQA (job-step related)	Job-oriented Fixed Protection key=0 Job-step-related Swappable Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire fixed, accountable, protected storage in the LSQA for the user's region that is job-oriented and freed when the job step terminates.
255	Space within LSQA (explicitly assigned and freed)	Job-oriented Fixed Protection key=0 Explicitly assigned and freed Swappable Not fetch-protected	Allows a task running in key 0 or supervisor state to acquire fixed, nonaccountable, protected storage in the LSQA that is job-oriented and must be explicitly freed.

Figure 6-1 (Part 3 of 3). Table showing attributes and uses of subpools

In addition to their use in distinguishing among the kinds of virtual storage, subpools give the user a means of sharing or isolating storage within his region. A user task and its subtasks operate within the same region, and since the hardware storage protection keys are assigned to the entire region, they cannot be used to isolate areas of storage to particular subtasks. Subpools 0-127 allow the user to isolate storage from another task executing with the same storage protection key, or from later entries into the same program. Unshared subpools 0-127 are freed when the user task terminates.

The user may also use subpools 0-127 to share common areas of storage between tasks of the same job step. Shared subpools are freed when the owning task terminates, not when a sharing, but not owning subtask terminates.

SYSTEM QUEUE AREA AFTER INITIALIZATION

The SQA (system queue area) contains system control blocks required from one IPL to another. The size of the SQA is always a 64K segment multiple. The number of 64K segments set aside for the SQA is a system generation option and can be changed during system initialization. After initialization, SQA virtual address space cannot expand; however, the number of page frames assigned to the SQA does increase and decrease while the system is running.

Because system control blocks are used so frequently, each SQA page is fixed in real storage when it is allocated. If the pages were not fixed and if (in the worst case), each control block in a queue were on a separate page, a search of that queue would cause excessive paging. Each reference to a control block could cause a page fault and a page-in operation to bring that page into real storage. To avoid this, each SQA page is fixed in real storage when it is allocated.

Before the system is generated, users of the VS2 system must specify the amount of space needed for the SQA and, optionally, the size of the quickcell area within the SQA. During execution of the nucleus initialization program, a DQE (descriptor queue element) containing a record of the number of bytes assigned to the SQA is built within the area. In addition to the DQE, an FQE (free queue element) is built that contains the number of bytes of available space (initially, all space is available) in the SQA. Optionally, the initialization program also builds a QCDBLK (quickcell descriptor block) describing SQA quickcells (see "Allocating and Freeing Quickcells" in this section). Location GOVRFLB in the nucleus (pointed to by the secondary CVT) contains a pointer to the descriptor queue element, and the descriptor queue element contains a pointer to the free queue element. Figure 6-2 shows the control blocks used for virtual storage supervision as they are immediately after system initialization.

NIP (the nucleus initialization program) initializes other control blocks in the SQA to describe the contents of the dynamic area. The dynamic area consists of the pageable dynamic area and the nonpageable dynamic area. The dynamic area is defined when NIP initializes the partition queue element for the dynamic area for pageable tasks (V=V PQE) and the partition queue element for the dynamic area for nonpage-

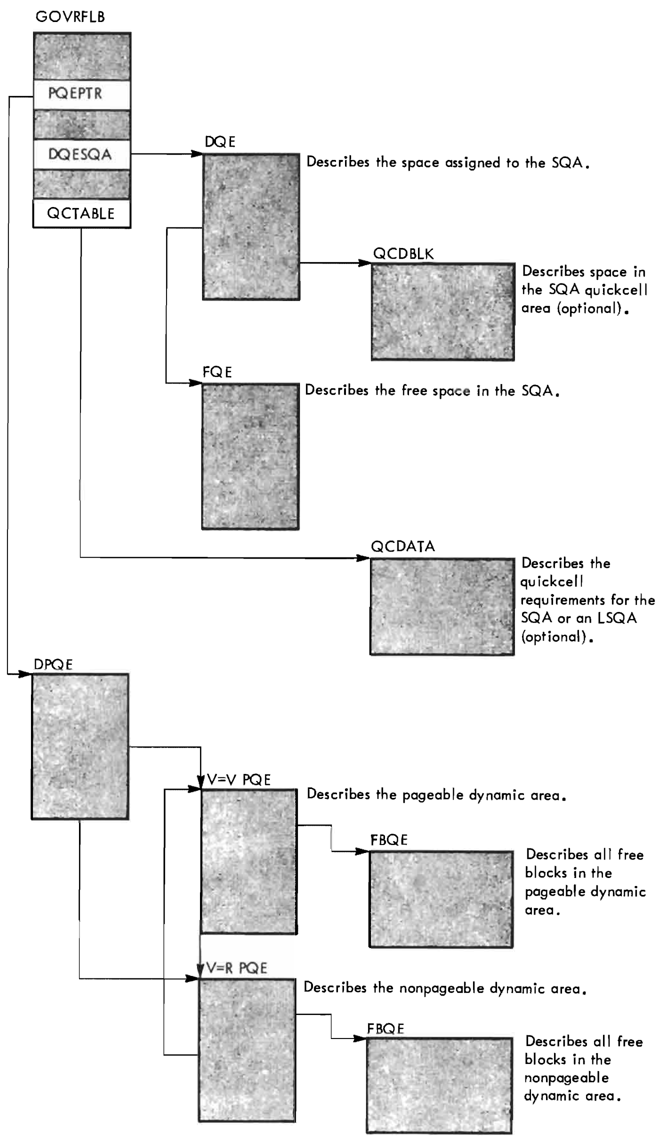

GOVRFLB
PQEPTR
DQESQA
QCTABLE
DQE
Describes the space assigned to the SQA.
QCDBLK
Describes space in the SQA quickcell area (optional).
FQE
Describes the free space in the SQA.
QCDATA
Describes the quickcell requirements for the SQA or an LSQA (optional).
DPQE
V=V PQE
Describes the pageable dynamic area.
FBQE
Describes all free blocks in the pageable dynamic area.
V=R PQE
Describes the nonpageable dynamic area.
FBQE
Describes all free blocks in the nonpageable dynamic area.

Figure 6-2. VSS control blocks after system initialization.

able tasks (V=R PQE). NIP anchors these PQEs to the PQEPTR field in the GOVRFLB control block in the nucleus via a dummy PQE.

The V=V PQE is the anchor for the free block queue that describes blocks of available space within the pageable dynamic area. After initialization, the pageable dynamic area is described by only one FBQE (free block queue element) since the entire area is available. When the system requests any of the following services, appropriate control block queues are modified to remove space from, or return space to, the free pageable dynamic area:

GET LSQA (subpool 249)

FREE LSQA (subpool 249)

GETPART V=V (subpool 247)

FREEPART V=V (subpool 247)

GETSWA (subpool 236)

FREESWA (subpool 236)

(Figure 6-6 illustrates virtual storage after space has been allocated for a pageable task.)

The V=R PQE is the anchor of the control block queue that describes blocks of available space within the nonpageable dynamic area. After initialization, the entire nonpageable area is available for allocation and is described by only one FBQE. When a system routine requests GETPART V=R or FREEPART V=R, appropriate control block queues are modified to allocate space from, or return space to, the free nonpageable dynamic area. (Figure 6-7 illustrates virtual storage after an LSQA and a region have been allocated for a nonpageable task.)

ALLOCATING SPACE IN THE SYSTEM QUEUE AREA

The system queue area is used by the supervisor to satisfy requests for space in subpools 241, 243, 244, and 245. Requests for space in subpools 233, 234, 235, 253, 254, and 255 are also satisfied from the SQA if there is not enough space available in the LSQA or when a task is being abnormally terminated and its LSQA space is exhausted. (See "Allocating Space in a Local System Queue Area" below.)

Subpool 243 is used for task-related SQA space that is freed automatically when the task terminates. Subpool 244 is used for job-step related SQA space that is freed automatically when the job step terminates. Subpool 245 is used for explicitly assigned SQA space and must be explicitly freed. (See "Allocating and Freeing Quickcells" in this section.) Subpool 241, which is included in VS2 for compatibility with VS1, is treated as subpool 245. (For a complete list of VS2 subpools, see Figure 6-1.)

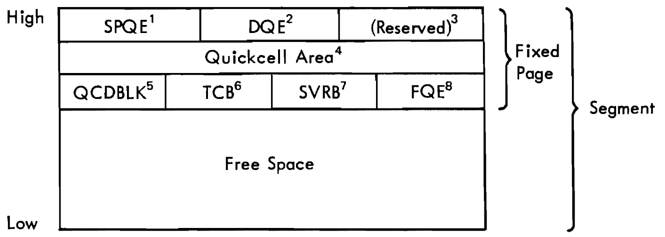
If all of the virtual space for the SQA is exhausted, or if a fixed page is not available for an SQA request, the system is placed in an enabled wait state.

ALLOCATING SPACE IN A LOCAL SYSTEM QUEUE AREA

An LSQA (local system queue area) consists of one or more segments of virtual storage. One LSQA is created for each initiator and lasts for the life of the task. The system uses each LSQA to contain various control blocks. VSS uses each LSQA to store queues of control blocks that contain the information necessary to keep track of the free space and allocated space within a user's region. Figure 6-3 shows the format of the LSQA segments after they have been initialized by the Get LSQA Segment routine.

Segments for an LSQA are assigned from the highest addressed segments available in the pageable dynamic area (see Figure 6-6 and Diagram 6.11). Within an LSQA, space is allocated from the highest addresses to the lowest.

Storing in the LSQA is restricted to control program routines that operate under a protection key 0. Non-key-0 users can only read LSQA storage.



Field Explanations:

[1] Subpool queue element (SPQE) for LSQA

[2] Descriptor queue element (DQE describes the entire LSQA segment)

[3] Reserved for future use

[4] Quickcell area (see "Quickcell Allocation")

[5] Quickcell descriptor block

[6] Skeleton TCB

[7] Supervisor request block (SVRB)

[8] Free queue element (FQE)

Figure 6-3. An LSQA segment after initialization.

To obtain space in an LSQA, requesters must specify subpools 233, 234, 235, 253, 254, or 255 in their GETMAIN macro instructions.

Normal LSQA requests for one page or less are not satisfied across page boundaries. LSQA segments are allocated from the pageable dynamic area. Pages within the segments must be obtained and fixed before a GETMAIN request for LSQA can be satisfied. If a page of real storage is not available, the requesting task is abnormally terminated and all requests for LSQA space by the ABEND routine are statisfied from the SQA. This is done to avoid an ABEND recursion (a second entry to the ABEND routine).

Normal Requests for LSQA Storage

Normal LSQA requests for a page or less that are not quickcell candidates are satisifed by searching the FQE (free queue element) queue in the LSQA to find a block of storage closest in size and large enough to satisfy the request. When an LSQA request is satisfied, a new FQE must be created to replace the FQE for the space just allocated (unless the request was for exactly the size of the free area described by an FQE, in which case that FQE is simply removed from the FQE queue and not replaced.)

Storage blocks in subpool 233, 234, 253 and 254 have an AQE (allocated queue element) appended to them. Subpool 233 and 253 AQEs are chained from the requesting task's AQE queue. Subpoool 234 and 254 AQEs are chained from the requesting task's job-step task AQE queue. The AQE contains the length of the allocated area (which includes the AQE itself).

No record is kept of the allocated space by the system for subpools 235 or 255.

Subpool 233, 243, and 253 Allocation

When subpoool 233, 243, or 253 is specified in a GETMAIN macro instruction, the GETMAIN routines add eight bytes to the size requested, and the location of the beginning of the available space is determined. A GETMAIN routine builds an AQE in the first eight bytes of the requested area. It then chains the AQE to an AQE queue whose origin is in the TCB (TCBAQE field) of the task for which the space was requested. When that task is terminated, the termination routines issue a FREEMAIN macro instruction to free all space in the subpool that had not been already freed.

Subpool 234, 244, and 254 Allocation

When subpool 234, 244, or 254 is specified in a GETMAIN macro instruction, a GETMAIN routine builds an allocated queue element as it does for subpoools 233, 243, and 253, but chains the AQE to an AQE queue whose origin is in the TCB of the job step for which the space was requested. When that job step is completed, the termination routines issue a FREEMAIN macro instruction to free all space in the subpool that had not been already freed.

Subpool 235, 241, 245 and 255 Allocation

When subpool 235, 241, 245, or 255 is specified in a GETMAIN macro instruction, no allocated queue element is built. It is the responsibility of the requester to ensure that the space is freed with a FREEMAIN macro instruction.

ALLOCATING AND FREEING PAGES IN A SYSTEM WORK AREA

SWA (system work area) segments are created, one per request, in the highest addressed segments available in the pageable dynamic area. The segments are created by the GETSWA routine when processing SVC 4 GETMAIN requests for subpool 236 (see Diagram 6.20).

The system requests SWA pages by issuing an SVC 10 GETMAIN for subpool 237. This causes the GETPAGE virtual storage supervision routine to allocate page-size blocks to the system from an existing SWA segment (see Diagram 6.21).

Allocation of space within the SWA is controlled by a SWAB (system work area block) which contains a bit map representing the 16 pages of the SWA segment. There is one SWAB for each SWA segment. The SWABs are chained on the SWAH (system work area header) from the most recently allocated SWAB to the oldest. The GETPAGE function allocates pages from the SWAs by searching the chain of SWABs (see Figure 6-4). If there are no pages left for allocation, the scheduler is notified by return code.

The system frees pages in the SWA when they are no longer needed by issuing an SVC 10 FREEMAIN for subpool 237. The FREEPAGE routine notifies the system whenever a segment of SWA has become completely free (see Diagram 6.39). The system frees SWA segments by issuing an SVC 10 FREEMAIN for subpool 236 (see Diagram 6.38).

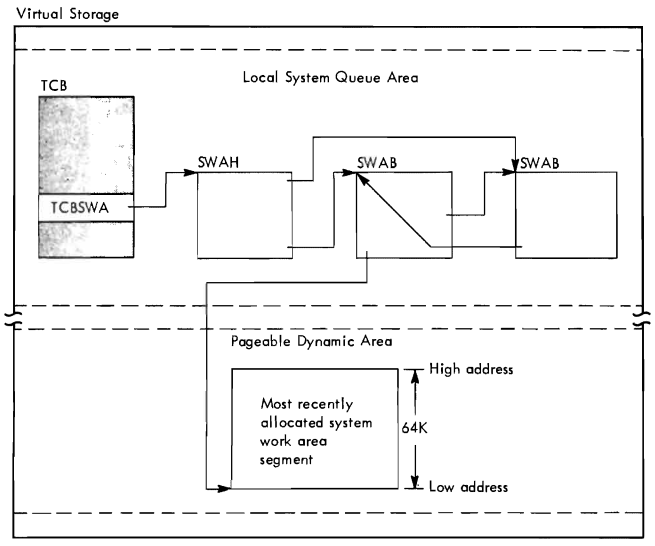


Figure 6-4. Control blocks used to allocate storage in the system work area.

ALLOCATING AND FREEING QUICKCELLS

(See Diagrams 6.12, 6.13, and 6.37)

Quickcells are small blocks of storage in the SQA (subpool 245) and the LSQAs (subpool 255) that can be allocated quickly and that are expected to be used over and over again for short periods of time.

A quickcell can range in size from 8 to 256 bytes, but must be a multiple of 8 bytes. The number and sizes of the quickcells are defined at system generation and can be changed during nucleus initialization.

The quickcell area in the SQA and in each LSQA is described by a QCDBLK (quickcell descriptor block) which contains a QCDE (quickcell descriptor entry) for each quickcell. The QCDBLB for SQA quickcells is built in the SQA during nucleus initialization. The QCDBLK for an LSQA quickcell area is built in the LSQA when the LSQA is allocated (see Diagram 6.12).

Quickcells can be allocated and released faster than normal SQA and LSQA storage. When GETMAIN and FREEMAIN routines process a request for a block of storage from subpool 245 or 255 that is explicitly requested from quickcells, the routines bypass the normal search and updating of the storage control blocks and use a special branch entry (QCBRANCH) to satisfy the request.

To allocate a quickcell, instead of searching the FQE queue, QCBRANCH indexes directly into the bit map in the appropriate QCDBLK to determine whether a quickcell is available to satisfy the request. If it finds a quickcell, it marks it allocated and passes its address to the caller. If a quickcell for the length requested is not available, the request is processed like a normal request for storage in the SQA or LSQA.

To free a quickcell, QCBRANCH simply finds the quickcell entry that matches the address to be freed and marks the entry available for allocation.

Each GETMAIN quickcell request must result in one FREEMAIN request for the whole amount and not two or three FREEMAIN requests, each for part of the space. For example, a request for 80 bytes must result in a FREEMAIN of 80 bytes and not a FREEMAIN of 40 and another of 40.

SUPERVISING EXTERNAL PAGE STORAGE

(See Diagrams 6.23, 6.24, and 6.40)

NIP (the nucleus initialization program) calculates the total amount of external page storage needed for the system and the number of external pages available for allocation. It places the count of all external pages available for allocation in the "total uncommitted" field and the "system uncommitted" field of the PVT (page vector table). The "total uncommitted" field contains a constant value while the "system uncommitted" field varies as external pages are allocated and freed.

NIP initializes the "batch committed" and "TSO committed" fields in the PVT to zero and sets the maximum TSO and maximum batch processing threshold counts as specified by the operator (see the OS/VS IPL and NIP Logic manual).

As batch jobs for V=V regions are initiated, the GETPART routine allocates the regions. GETPART assigns a virtual storage region for the job and ensures that there is an external page to back up each virtual page (that is, if the virtual region size for the batch job is 128K, there must be 32 pages of external page storage available). The GETPART routine decrements the "system uncommitted" count and increments the "batch committed" count by the number of pages assigned to the region (see Diagrams 6.23 and 6.40).

For TSO tasks, the external page storage counters are incremented by a percentage of the amount of virtual storage assigned. A logon buffer is used to predict whether sufficient external pages are available to allow a new TSO user to enter the system.

If not enough external pages are available, no new TSO jobs are initialized.

Batch jobs, once initiated, should not run out of external pages since they are guaranteed 100 per cent back-up of virtual storage with external page storage. However, only a fraction of the TSO user's virtual storage requirement is backed up by external page storage. Consequently, a count is maintained of the pages used by TSO users. If time sharing jobs exceed the amount of external page storage that can be used, they are terminated.

GETPART reserves an area in external page storage to guarantee that a TSO task can be swapped out to external page storage. This TSO buffer is large enough to accommodate any TSO task in real storage that may be forced to log off, thus ensuring that the swap-out operation will not require use of external page storage that is committed to batch jobs (see Diagrams 6.24 and 6.40).

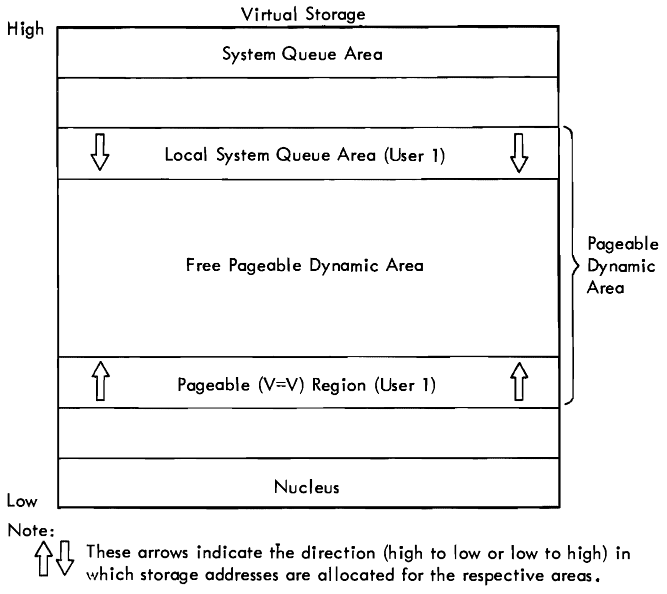

Virtual Storage
High
System Queue Area
Local System Queue Area (User 1)
Free Pageable Dynamic Area
Pageable (V=V) Region (User 1)
Pageable Dynamic Area
Nucleus
Low
Note:
These arrows indicate the direction (high to low or low to high) in which storage addresses are allocated for the respective areas.

Figure 6-5. Virtual storage after allocation of an LSQA and region for one pageable (V=V) task.

ALLOCATING SPACE FOR A REGION

(See Diagrams 6.14 through 6.19)

When an initiator issues a GETMAIN macro instruction for subpool 242 or 247, storage is obtained from the area represented by either the V=R PQE or V=V PQE respectively. A new PQE and an FBQE are created in the LSQA for the new region. The address of the PQE is inserted in the job step's TCB.

The routine that obtains regions upon request is the GETPART routine. The GETPART routine is invoked by either a register-form GETMAIN (SVC 10) request (always an unconditional single-region request) or a list-form GETMAIN (SVC 4) ECB-mode request. The ECB-mode GETPART request applies to the conditional form of single-region and split-region list requests. An ECB is contained in the requester's JSCB (job step control block). This type of request enables a job to be canceled when it cannot be initialized because sufficient storage space is not available. The ECB-mode GETPART request must be used when requesting space for nonpageable region or for a specific (checkpoint/restart) pageable region.

Space for regions is obtained from the dynamic area of virtual storage. (See figures 6-5 and 6-6.) The PQEPTR field at offset 8 in location GOVRFLB contains the address of a two-word DPQE (dummy partition queue element) less eight bytes. The first word of the DPQE contains the address of the V=R PQE (partition queue element). The second word of the DPQE contains the address of the V=V PQE (see Figure 6-2).

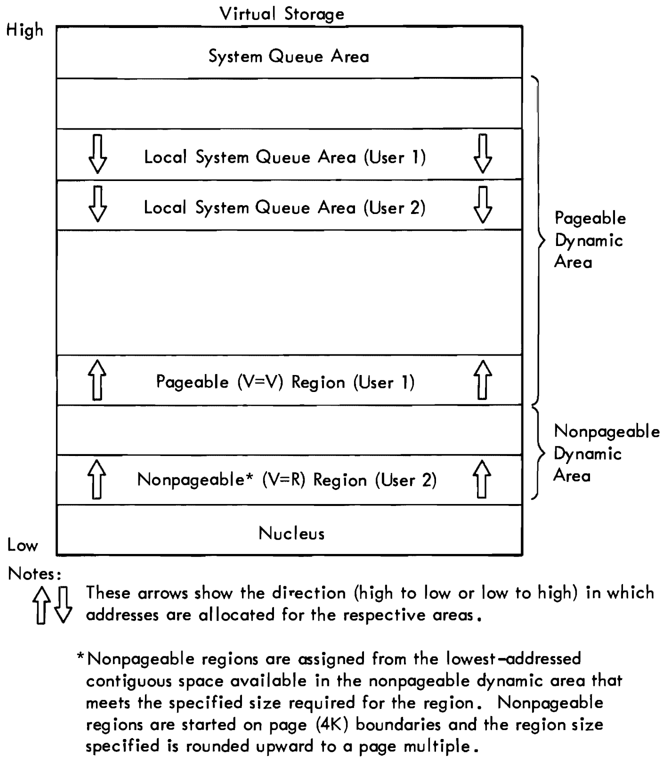

Virtual Storage
High
System Queue Area
Local System Queue Area (User 1)
Local System Queue Area (User 2)
Pageable (V=V) Region (User 1)
Nonpageable* (V=R) Region (User 2)
Pageable Dynamic Area
Nonpageable Dynamic Area
Nucleus
Low
Notes:
These arrows show the direction (high to low or low to high) in which addresses are allocated for the respective areas.
*Nonpageable regions are assigned from the lowest-addressed contiguous space available in the nonpageable dynamic area that meets the specified size required for the region. Nonpageable regions are started on page (4K) boundaries and the region size specified is rounded upward to a page multiple.

Figure 6-6. Virtual storage after allocation of LSQAs and regions for one pageable task and one nonpageable task.

The first and second words of the PQE point to the first and last FBQEs (free block queue elements) associated with the area of storage described by the PQE.

FBQEs contain a count of the number of free bytes available in that block of storage. FBQEs are chained forward and backward in address order so that virtual storage supervision routines can scan them from either high-to-low address or low-to-high address.

Before allocating a region, a VSS routine (CKTHRESH) checks whether the paging supervisor has sufficient resources available to service the size region requested. If not, control is returned to the caller with the request unsatisfied.

To assign a nonspecific region, the GETPART routine searches for the lowest block of free storage in the dynamic area large enough to satisfy the request. For a specific region request, it searches for the block that includes the region requested.

For each region, the GETPART routine builds an FBQE, a DPQE, and a PQE in the requester's local system queue area. It places in the FBQE a count of the contiguous free bytes that can be allocated in the region. It puts the address of the FBQE in the PQE and the address of the PQE in both fields of the DPQE (see Figure 6-8).

The GETPART routine also places the size of the region and the region address in the PQE and places the address of the dummy PQE minus eight bytes in the TCBPQE field of the requester's job-step TCB. The GETPART routine also establishes subpool 252 when a nonspecific region is obtained, to reserve a minimum of 4K minus 8 bytes, so that programs in supervisor state and protection key 0 can obtain storage in the region.

Region Reference-Validity Map

All user virtual storage regions have the same storage protection key. Storage protection between regions is provided by segment invalidation. Channel references, however, are not protected by segment invalidation; therefore, any channel reference requested by a non-key-0 program must be checked to ensure that the reference is valid. For this purpose, a table is built by the GETPART routine to define those virtual addresses that may be referenced by a non-key-0 program in a virtual storage region. This table is called the region reference-validity map.

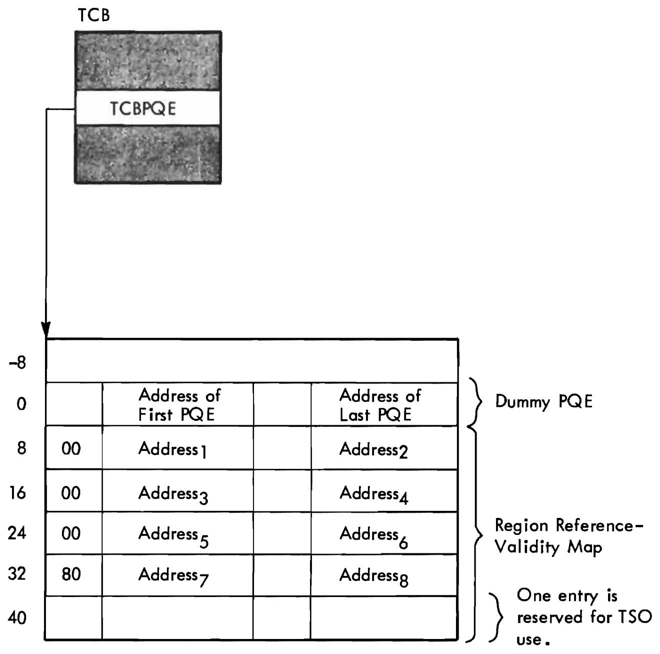
The region reference-validity map is constructed at the end of the dummy PQE when the GETPART routine allocates a region. The map consists of eight-byte elements, each containing the starting and ending address of a portion of storage that may be referenced by a task in the associated region. The addresses are placed in the map in ascending order. The storage described by the elements in the map are the nucleus, region, LSQA, and the area from the beginning of the LPA directory to the last addressable byte of virtual storage (the high address of the SQA). These storage areas are the only ones that may be referenced by the task.

Figure 6-7 illustrates the region reference-validity map.

ALLOCATING SPACE WITHIN A REGION

Any GETMAIN macro instruction in which subpools 0-127, 239, 240, 250, 251, or 252 are specified indicates that space within an existing region is desired.

Supervision of storage within a region as accomplished by using control block queues that originate from two fields in the TCB (task control block). TCBMSS points to the SPQE (subpool queue element) queue, which describes allocated areas of storage within a region. TCBPQE points to the DPQE (dummy partition queue element) which points to the PQE (partition queue element) queue which, with the FBQEs (free block queue elements), describes unallocated areas of storage within a region.



Note: The addresses in the region reference-validity map are arranged sequentially from the lowest address to the highest and they all start on segment boundaries.

Figure 6-7. Format of the region reference-validity map.

Management of Unallocated Storage Within a Region

There is one PQE per region. It is created when space for the region is allocated. Each PQE contains pointers to both ends of a two-way chain of FBQEs (free block queue elements) which describe 4K blocks (pages) of available storage space within a region. There may be any number of FBQEs for one region. Each FBQE contains three pointers, one to the free area that FBQE describes, one to the next-higher FBQE on the chain, and one to the next-lower FBQE. Each FBQE also includes a field that contains the number of bytes of free storage it represents. This count field is always a multiple of 4K.

Immediately after a region has been created, there is only one FBQE representing all the available space. All subsequent GETMAIN requests for subpool numbers issued within this region are satisfied from the space allocated for the region. The space is taken from that described by the FBQE for the region and is assigned the subpool number identified in the GETMAIN macro instruction.

Figure 6-8 illustrates the control block queues used to manage unallocated storage in a region.

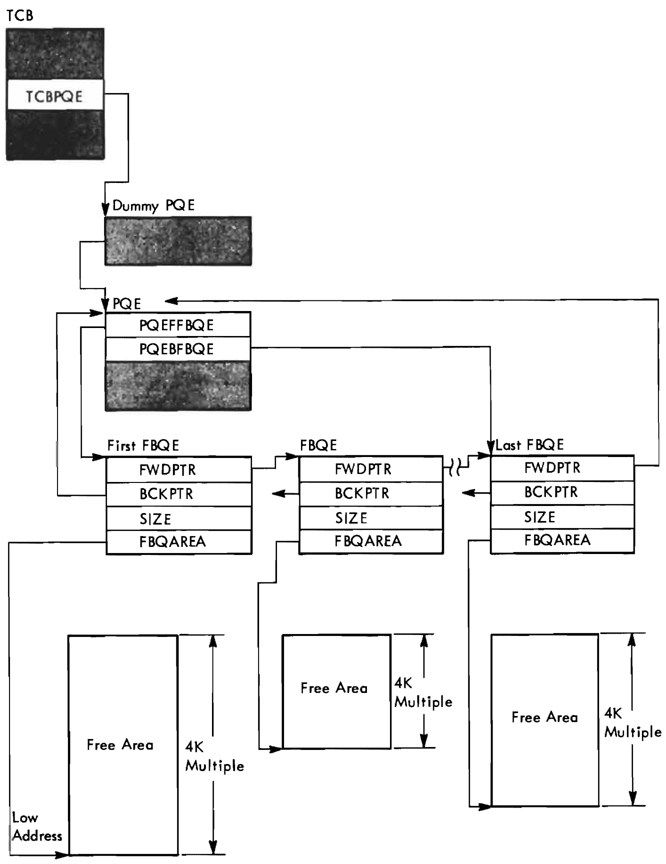


Figure 6-8. Control blocks used to manage unallocated storage within a region.

Supervision of Allocated Space Within a Region

When a GETMAIN macro instruction is executed, the VSS routines search the list of SPQEs (subpool queue elements) whose origin is in the TCB field TCBMSS (see Figure 6-9). Each SPQE anchors the control blocks that describe the space associated with an existing subpool number. If an SPQE is not found for the subpool requested, a new SPQE is created (see Diagram 6.5).

Each SPQE contains an identifying number (see Figure 6-3), control information that indicates whether the subpool is owned exclusively by this task or is being shared with another, and two pointers. These pointers create two chains. One chain is composed of other SPQEs representing other subpools. The other chain is a series of DQEs (descriptor queue elements) which describe blocks of storage within the subpool. Each DQE describes one or more contiguous 4K blocks of storage.

When a processing program issues a GETMAIN request, a series of 4K blocks large enough to satisfy the request are allocated to the subpool for which the request was made. From the amount of 4K blocks of storage so obtained, the number of bytes asked for in the GETMAIN instruction is subtracted. The remaining space (if any) is still available to be allocated to satisfy another GETMAIN request for the same subpool by the same task, or by a task that is sharing the same subpool. A FQE (free queue element) is used to describe the unallocated (and still available) storage within the space defined by a DQE. (See Figure 6-9 for an illustration of these queues.)

Processing the Initial Request for Subpool Space

Each time a request for space is received, the chain of SPQEs is scanned by the GETMAIN routines to determine whether the requested subpool exists. When the initial request for a particular subpool is received, the GETMAIN routines build a SPQE (subpool queue element) in the local system queue area. The SPQE contains the subpool number and, if other subpools exist, a pointer to another SPQE. (See Diagram 6.5.)

The GETMAIN routines also build a DQE (descriptor queue element) in the local system queue area, and place the address of the DQE in the subpool queue element. The

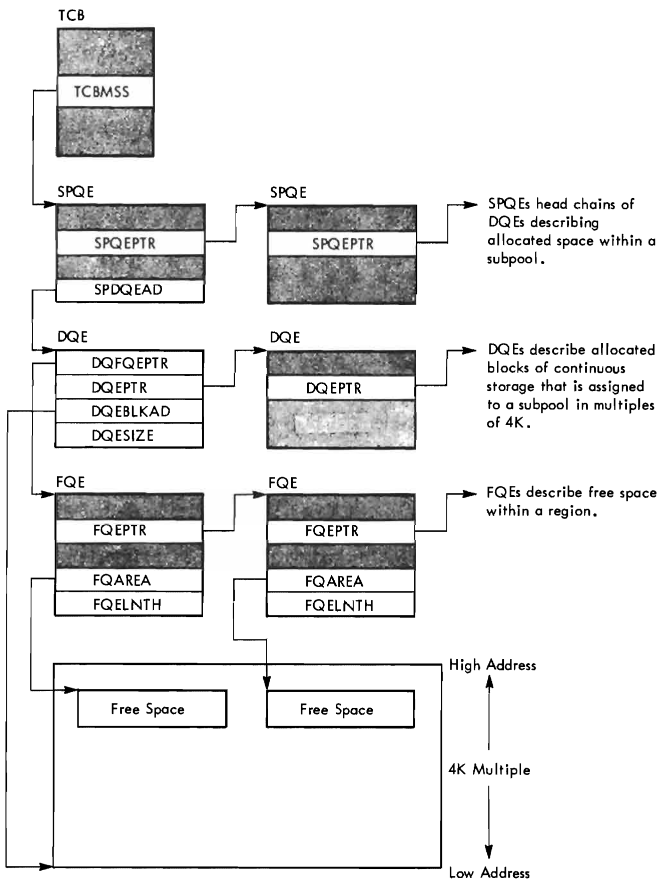

TCB
TCBMSS
SPQE
SPQEPTR
SPDQEAD
SPQE
SPQEPTR
SPQEs head chains of DQEs describing allocated space within a subpool.
DQE
DQFQEPTR
DQEPTR
DQEBLKAD
DQESIZE
DQE
DQEPTR
DQEs describe allocated blocks of continuous storage that is assigned to a subpool in multiples of 4K.
FQE
FQEPTR
FQAREA
FQELNTH
FQE
FQEPTR
FQAREA
FQELNTH
FQEs describe free space within a region.
Free Space
Free Space
High Address
4K Multiple
Low Address

Figure 6-9. Control blocks used to supervise allocated storage within a region.

DQE contains a count of the number of bytes of storage allocated to a block in the subpool (space within regions is assigned to subpools in multiples of 4K-byte blocks).

If any free space exists within the 4K-byte blocks defined by a DQE, a GETMAIN routine builds an FQE (free queue element) within the LSQA and places in it a count of the number of bytes available and the address of the highest byte +1 of the free space. All such FQEs for one contiguous area are chained together. The address of the first such FQE is placed into the associated DQE.

Processing Subsequent Requests for Subpool Space

When a GETMAIN request is issued for an already existing subpool (identified by an SPQE), a check is made to see if a DQE exists. (If a FREEMAIN had been issued for the entire subpool, an SPQE would remain, but there would be no DQE.) If there is no DQE, a new one is built as described in the preceding topic "Processing the Initial Request for Subpool Space". If there is a DQE, the FQE queue associated with the subpool is searched (see Diagram 6.7) and the request is satisfied from the free space within the subpool, if possible. If the request cannot be satisfied in this way, additional space is allocated (in multiples of 4K-byte blocks) to the subpool. The space is obtained from the free area described by FBQEs. The amount of storage obtained is represented by a DQE and one (or more) FQEs within the specified subpool. The number of bytes allocated is subtracted from the FBQE for the area from which storage was obtained, and the FBQE pointer is relocated if necessary. All DQEs representing space in the same subpool are chained together.

For nonpageable tasks, the storage protection key of the requesting task is assigned to the allocated storage and the storage is fetch-protected as each 4K-byte block is assigned. Then, when each block is freed, its storage protection key is reset to zero.

For pageable regions, the protection key and fetch protection are set in the external page table as each 4K-byte block is assigned, and the block is marked assigned to GETMAIN. Then, when the block is paged into real storage, the page is assigned the storage protection key and fetch protection by the paging supervisor. When a page is freed, it is released from both real and external page storage and an indication that the page no longer exists is set in the page table.

After space is assigned, storage usage information is recorded (see "Recording Storage Useage Information" below). Finally, the GETMAIN routines place the address of the assigned space into register 1 if an SVC 10 instruction caused entry, or place the address into the location specified by the requester if an SVC 4 instruction caused entry.

Processing if the Requested Space is not Available

If there is not enough free space in the region to satisfy a request, the CDPURGE routine attempts to purge one or more unused modules in the requester's region (see Diagram 6.9). The space freed by this purge may be sufficient to satisfy the current request.

If the purge flag (TCBJPQF) is set in the TCBJPQ field of the job-step TCB, the CDPURGE routine examines all CDEs (contents directory entries) in the job pack queue. Each CDE that has the "release" flag (CDREL) set in its attributes field represents a module in the region that is no longer needed (that is, there are no outstanding requests for the module by any routine in the job step.) For each such module, the CDPURGE routine branches to the CDDESTRY routine (in CDEXIT) to dequeue the CDE and free the associated module and its extent list.

After all CDEs in the region's job pack queue have been examined and all unused modules purged, and if the module purge has freed enough space to satisfy the request, the GETMAIN routines allocate the needed space to the requester's task, then return control to the requester.

RECORDING STORAGE USAGE INFORMATION

(See Diagram 6.10.)

The SMF Storage routine checks to determine whether an initiator has indicated that SMF recording should be performed, and then checks the TCB for the address of the TCT (timing control table). If there is no TCT, SMF storage information is not being recorded for this task. If there is a TCT, the SMF Storage routine performs the following functions for subpools 0-127 and 250-252:

- It determines whether the newly allocated storage exceeds either the "low-water mark" (LWM) or the "high-water mark" (HWM) for the region. The LWM is the address of the highest storage address allocated from the bottom of the region, and the HWM is the address of the lowest storage address allocated from the top of the region. If either is exceeded, the SMF Storage routine stores a new value in the TCT.
- It calculates the difference, in terms of 4K-byte blocks, between the LWM and HWM. A record of the minimum value for this difference is kept in the TCT. If the new allocation or release creates a new minimum, the SMF Storage routine records the new minimum difference in the TCT.

FREEMAIN ROUTINES

(See Diagram 6.27.)

The FREEMAIN routines service the FREEMAIN macro instruction, which is used to free space when it is no longer needed. Space assigned to a region, space within a region, space in a system work area, space in a local system queue area, or space in the system queue area may be freed. Basically, the FREEMAIN routines combine space being freed with existing free space or, if the adjacent space is not free, build new control blocks and add them to the appropriate queues of free blocks.

Freeing the Entire Space Assigned to a Region

The FREEPART function is invoked by a register-type FREEMAIN request (SVC10) or by a branch to RMBRANCH to release space from subpool 242 (V=R) or subpool 247 (V=V).

To free a region, the TCBPQE field of the TCB that represents the task for which the region is being used is checked to determine the address of the partition queue element for the region. The dummy PQE and region reference-validity map are cleared first. Then the space occupied by the PQE is released (see "Freeing Space in an LSQA or the SQA" below). Next, if the region to be freed is adjacent to an existing free area, it is combined with that area. This is done by adding the number of bytes in the region being freed to the size field of the free block queue element for the existing free area and, if necessary, changing the FBQE to point to the beginning of the newly enlarged free area.

If a region being freed is not adjacent to a free area, the FREEMAIN routines build an FBQE for the area and add it to the chain of FBQEs that represents all space available for allocation as regions.

The FREEPART routine also frees the eight bytes in subpool 252 obtained by the GETPART routine.

Freeing Space Within a Region

(See Diagram 6.29.)

To free space within a region, the SPQE (subpool queue element) representing the subpool from which space is to be freed is located. The address of the SPQE queue is in the TCBMSS field of the task control block for the task to which the space is assigned. The DQE (descriptor queue element) that represents the area in which the space is to be freed is located. Next, the two FQEs (free queue elements) between which the space exists are located and a new FQE is constructed to represent the newly freed space. This FQE is either added to the chain of FQEs or, if the space lies adjacent to another free area, is combined with the FQE of the adjacent free area (see Diagram 6.35).

A test is then made to determine whether the resulting free area contains any free 4K-byte blocks that begin on a page boundary. If it does, and the block is adjacent to an existing free 4K-byte block, the number of bytes to be freed is added to the count field in the FBQE representing the existing free space and the space is cleared. (See Diagram 6.34.)

For a nonpageable task, the protection keys of the released blocks are set to zero, with fetch protection on. For pageable tasks, the page table entries for the released blocks are marked not assigned and the storage key in the external page table entries are set to zero.

If the block being freed is not adjacent to a free 4K-byte block, a new FBQE is constructed and paging supervisor references to the pages are deleted. The number-of-bytes count in the appropriate DQE is then decremented to reflect the storage being removed from the subpool. When this count reaches zero, the DQE is eliminated. A new DQE is created and the old one is updated when space in the middle of that described by the old DQE is freed.

The SMF Storage routine (FMSMFCRE) is used to maintain storage usage information in the TCT (timing control table). (See "Recording Storage Usage Information" earlier in this section.)

<u>Freeing Space in an LSQA or the SQA</u>

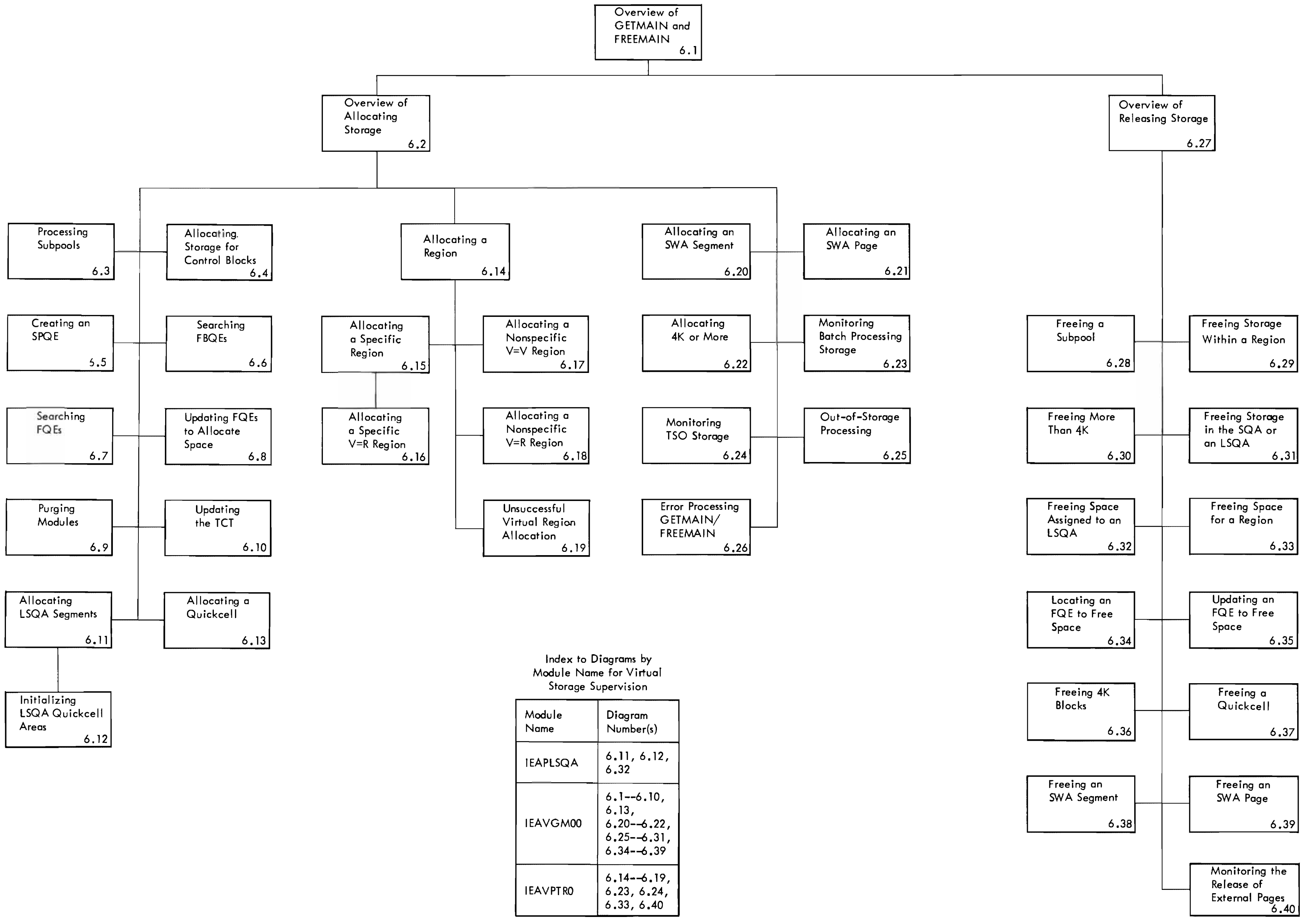
(See Diagram 6.31.)

When processing requests to allocate storage in the SQA or an LSQA, fixed real storage is obtained before the request is satisfied. A request of 4K or less is not satisfied across page boundaries. When a preciously allocated SQA or LSQA page is freed, the real page assigned to it is released.

To free space in the SQA or an LSQA, a FREEMAIN routine first locates the DQE (descriptor queue element) that represents the space. It next checks to determine wether space within subpcols 233, 234, 243, 244, 253, or 254 is to be freed. If so, eight bytes are added to the size of the area to be freed (to include the AQE that is contained in the area), and eight bytes are subtracted from the address of the space.

For subpool 233, 234, 243, 244, 253, and 254, the address of the AQE is obtained from the TCBAQE field of the associated TCB. If the entire area defined by the AQE is to be freed, the AQE is simply removed from the AQE queue. If the lower boundary is being freed and the upper boundary is not, or if space in-between is being freed, a new AQE is created for the remaining storage. Otherwise, the byte count of the original AQE is reduced by the amount of storage being freed. Any resulting contiguous free areas are combined by combining FQEs.

• Diagram 6.0
Virtual Storage Supervision
Visual Table of Contents

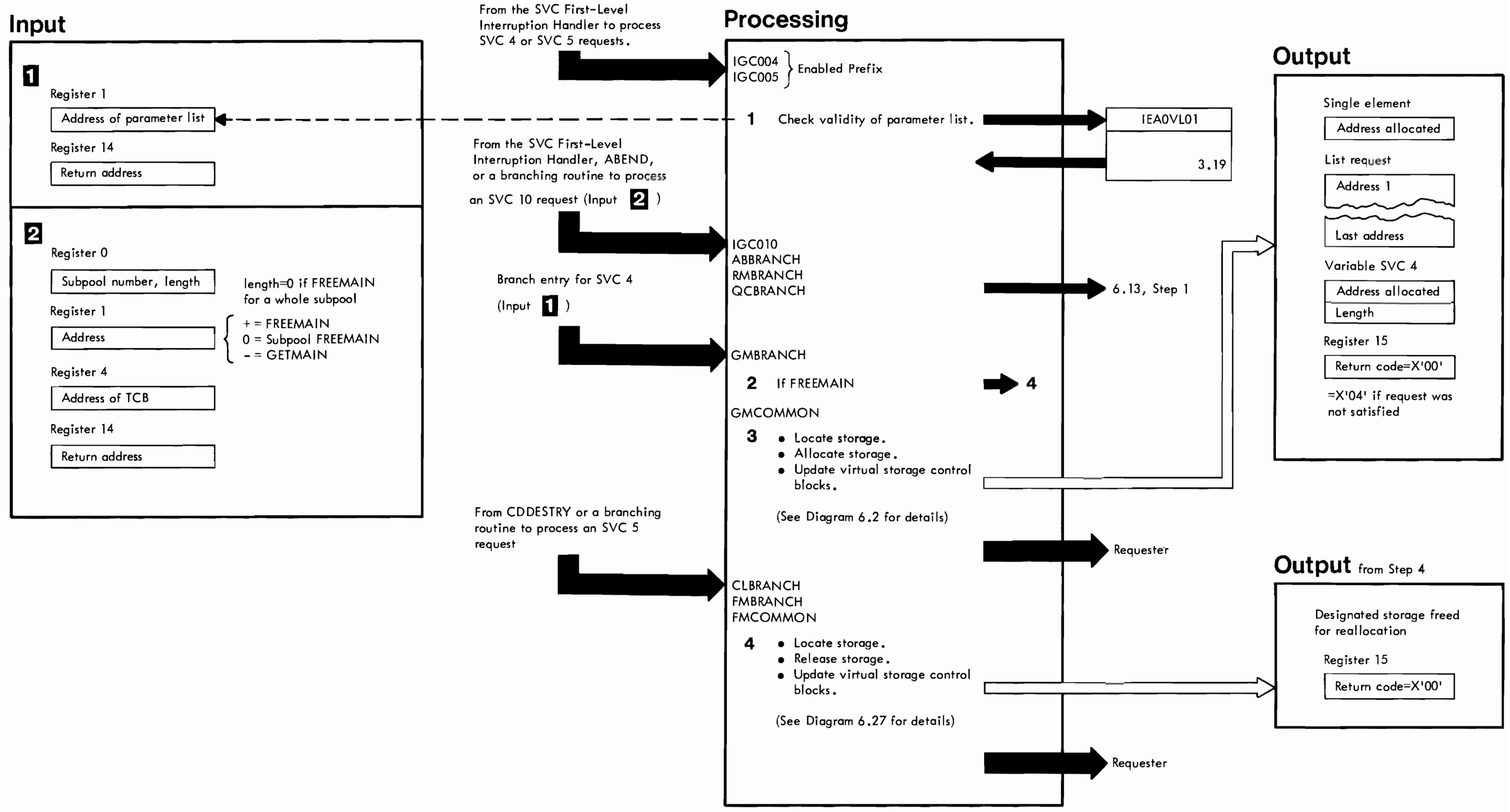


Overview of GETMAIN and FREEMAIN 6.1
Overview of Allocating Storage 6.2
Overview of Releasing Storage 6.27
Processing Subpools 6.3
Allocating Storage for Control Blocks 6.4
Creating an SPQE 6.5
Searching FBQEs 6.6
Searching FQEs 6.7
Updating FQEs to Allocate Space 6.8
Purging Modules 6.9
Updating the TCT 6.10
Allocating LSQA Segments 6.11
Allocating a Quickcell 6.13
Initializing LSQA Quickcell Areas 6.12
Allocating a Region 6.14
Allocating a Specific Region 6.15
Allocating a Nonspecific V=V Region 6.17
Allocating a Specific V=R Region 6.16
Allocating a Nonspecific V=R Region 6.18
Unsuccessful Virtual Region Allocation 6.19
Allocating an SWA Segment 6.20
Allocating an SWA Page 6.21
Allocating 4K or More 6.22
Monitoring Batch Processing Storage 6.23
Monitoring TSO Storage 6.24
Out-of-Storage Processing 6.25
Error Processing GETMAIN/ FREEMAIN 6.26
Freeing a Subpool 6.28
Freeing Storage Within a Region 6.29
Freeing More Than 4K 6.30
Freeing Storage in the SQA or an LSQA 6.31
Freeing Space Assigned to an LSQA 6.32
Freeing Space for a Region 6.33
Locating an FQE to Free Space 6.34
Updating an FQE to Free Space 6.35
Freeing 4K Blocks 6.36
Freeing a Quickcell 6.37
Freeing an SWA Segment 6.38
Freeing an SWA Page 6.39
Monitoring the Release of External Pages 6.40

Index to Diagrams by Module Name for Virtual Storage Supervision

Module Name	Diagram Number(s)
IEAPLSQA	6.11, 6.12, 6.32
IEAVGM00	6.1--6.10, 6.13, 6.20--6.22, 6.25--6.31, 6.34--6.39
IEAVPTR0	6.14--6.19, 6.23, 6.24, 6.33, 6.40

Diagram 6.1
Overview of GETMAIN and FREEMAIN


Input
1
Register 1
Address of parameter list
Register 14
Return address
2
Register 0
Subpool number, length
length=0 if FREEMAIN for a whole subpool
Register 1
Address
+ = FREEMAIN
0 = Subpool FREEMAIN
- = GETMAIN
Register 4
Address of TCB
Register 14
Return address
From the SVC First-Level Interruption Handler to process SVC 4 or SVC 5 requests.
From the SVC First-Level Interruption Handler, ABEND, or a branching routine to process an SVC 10 request (Input 2)
Branch entry for SVC 4 (Input 1)
From CDDESTRY or a branching routine to process an SVC 5 request
Processing
IGC004
IGC005
Enabled Prefix
1 Check validity of parameter list.
IEAOVL01
3.19
IGC010
ABBRANCH
RMBRANCH
QCBRANCH
6.13, Step 1
GMBRANCH
2 If FREEMAIN
4
GMCOMMON
3 • Locate storage.
• Allocate storage.
• Update virtual storage control blocks.
(See Diagram 6.2 for details)
Requester
CLBRANCH
FMBRANCH
FMCOMMON
4 • Locate storage.
• Release storage.
• Update virtual storage control blocks.
(See Diagram 6.27 for details)
Requester
Output
Single element
Address allocated
List request
Address 1
Last address
Variable SVC 4
Address allocated
Length
Register 15
Return code=X'00'
=X'04' if request was not satisfied
Output from Step 4
Designated storage freed for reallocation
Register 15
Return code=X'00'

1 Diagram 6.1. Overview of GETMAIN and FREEMAIN (Module IEAVGM00)

INPUT

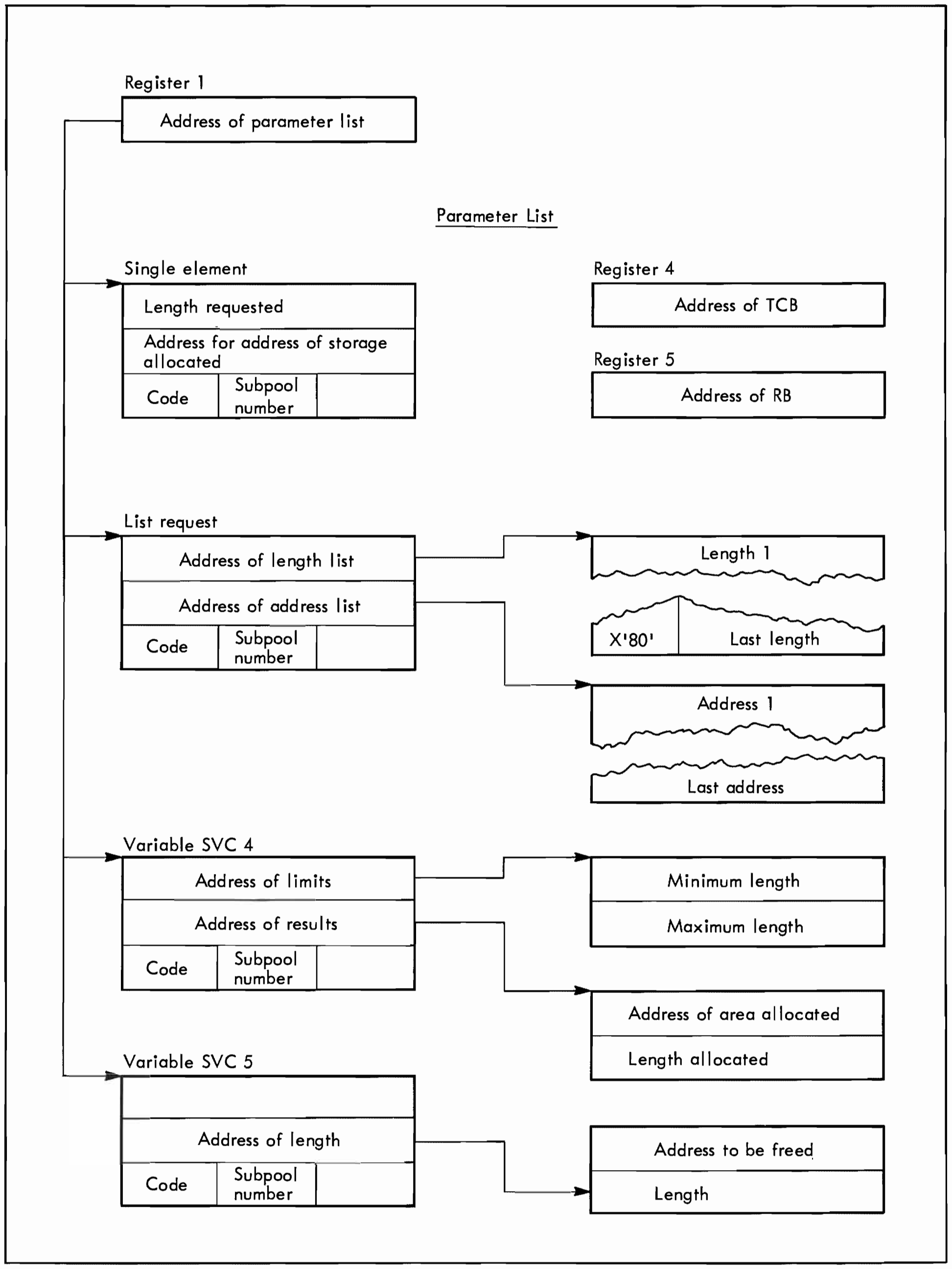
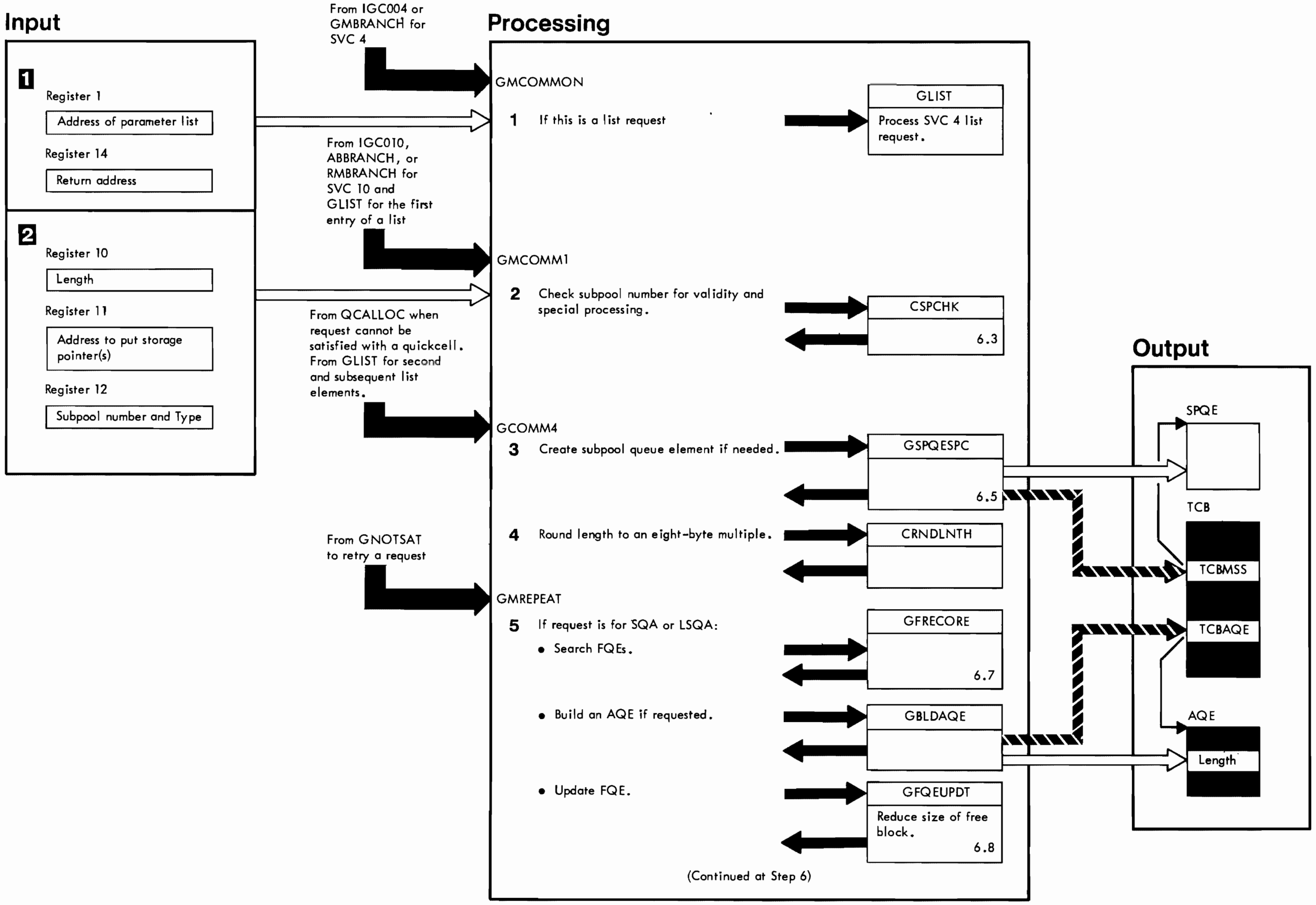


Diagram 6.2 (Steps 1-5)
Overview of Allocating Storage


Input
1
Register 1
Address of parameter list
Register 14
Return address
2
Register 10
Length
Register 11
Address to put storage pointer(s)
Register 12
Subpool number and Type
From IGC004 or GMBRANCH for SVC 4
From IGC010, ABBRANCH, or RMBRANCH for SVC 10 and GLIST for the first entry of a list
From QCALLOC when request cannot be satisfied with a quickcell. From GLIST for second and subsequent list elements.
From GNOTSAT to retry a request
Processing
GMCOMMON
GLIST
Process SVC 4 list request.
GMCOMM1
CSPCHK
6.3
GCOMM4
GSPQESPC
6.5
CRNDLNTH
GMREPEAT
GFRECORE
6.7
GBLDAQE
GFQEUPDT
Reduce size of free block.
6.8
Output
SPQE
TCB
TCBMSS
TCBAQE
AQE
Length

1 If this is a list request

2 Check subpool number for validity and special processing.

3 Create subpool queue element if needed.

4 Round length to an eight-byte multiple.

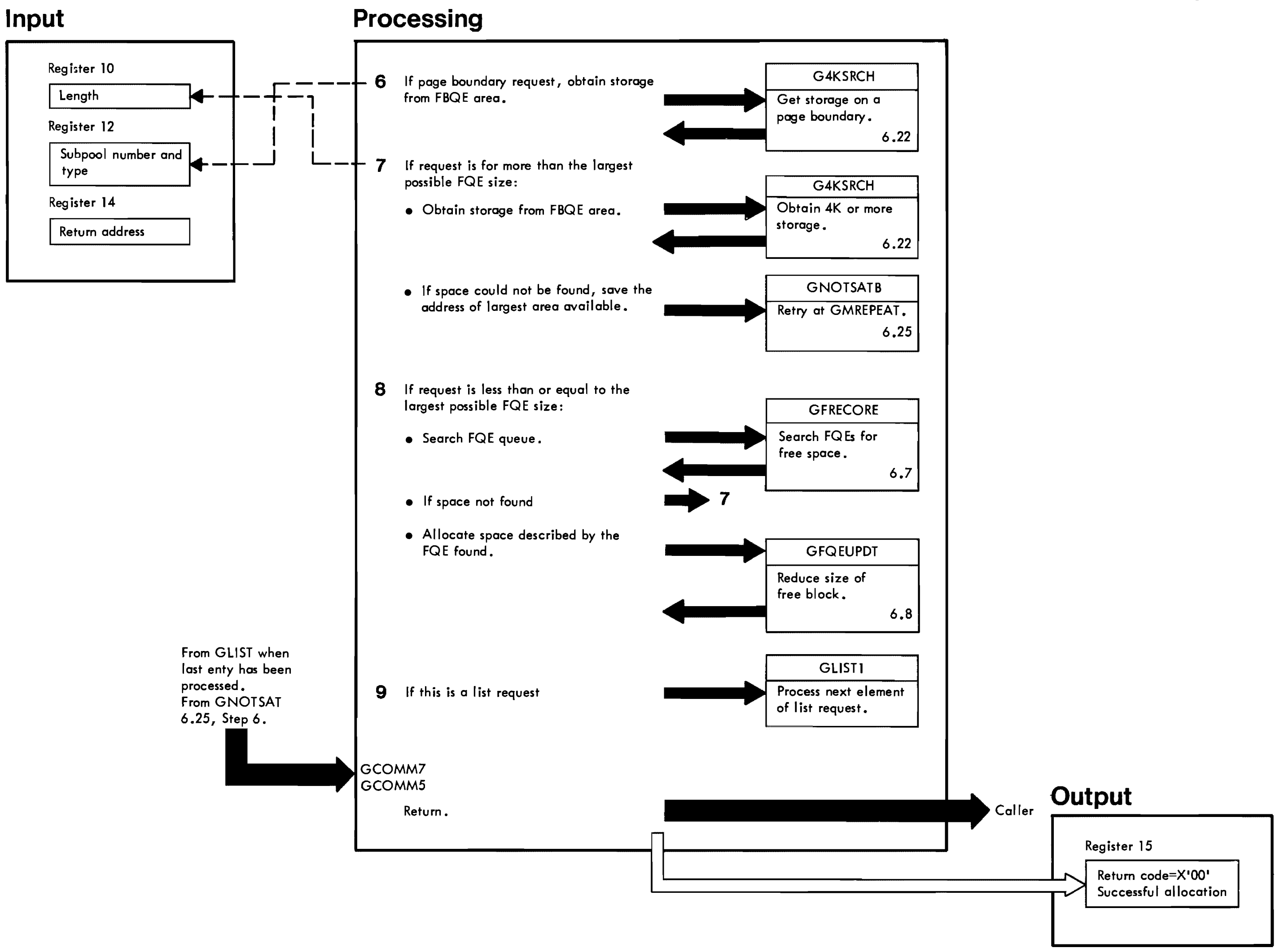
5 If request is for SQA or LSQA:
- Search FQEs.
- Build an AQE if requested.
- Update FQE.

(Continued at Step 6)

Diagram 6.2 (Steps 1-5) Overview of Allocating Storage (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 The parameter list is shown in the notes for Diagram 6.1. These are the input parameters for the following entry points: GMCOMMON, GCOMM4, GMREPEAT, and GCOMM5.		
2 These are the input parameters for GMCOMM1.		
1 The GLIST routine is effectively a part of the GETMAIN Common routine. The first time GLIST is called, it enters the GETMAIN Common routine at GMCOMM1. When the Common routine successfully satisfies a list request, it gives control to GLIST1 (see Step 9). All successive entries to the Common routine from GLIST1 are made at GCOMM4 to eliminate the subpool check. When all entries have been processed, the Common routine is entered at GCOMM5 to return to the caller.	GMCOMMON	GMCOMM1 GLIST
2	GMCOMMON GSPCHK	GMCOMM4
3 GSPCHK returns the address of the SPQE if one exists. If there is none, the GETMAIN Common routine branches to GSPQESPC to create one.	GSPQESPC	
5 The GETMAIN Common routine branches to GELDAQE to build an allocated queue element for subpool 243, 244, 253, and 254 requests.	GELDAQE	

Diagram 6.2 (Steps 6-10)
Overview of Allocating Storage



Input

Register 10
Length
Register 12
Subpool number and type
Register 14
Return address

Processing

6 If page boundary request, obtain storage from FBQE area.

G4KSRCH
Get storage on a page boundary.
6.22

7 If request is for more than the largest possible FQE size:

- Obtain storage from FBQE area.

G4KSRCH
Obtain 4K or more storage.
6.22

- If space could not be found, save the address of largest area available.

GNOTSATB
Retry at GMREPEAT.
6.25

8 If request is less than or equal to the largest possible FQE size:

- Search FQE queue.

GFRECORE
Search FQEs for free space.
6.7

- If space not found → 7

- Allocate space described by the FQE found.

GFQEUPDT
Reduce size of free block.
6.8

9 If this is a list request

GLIST1
Process next element of list request.

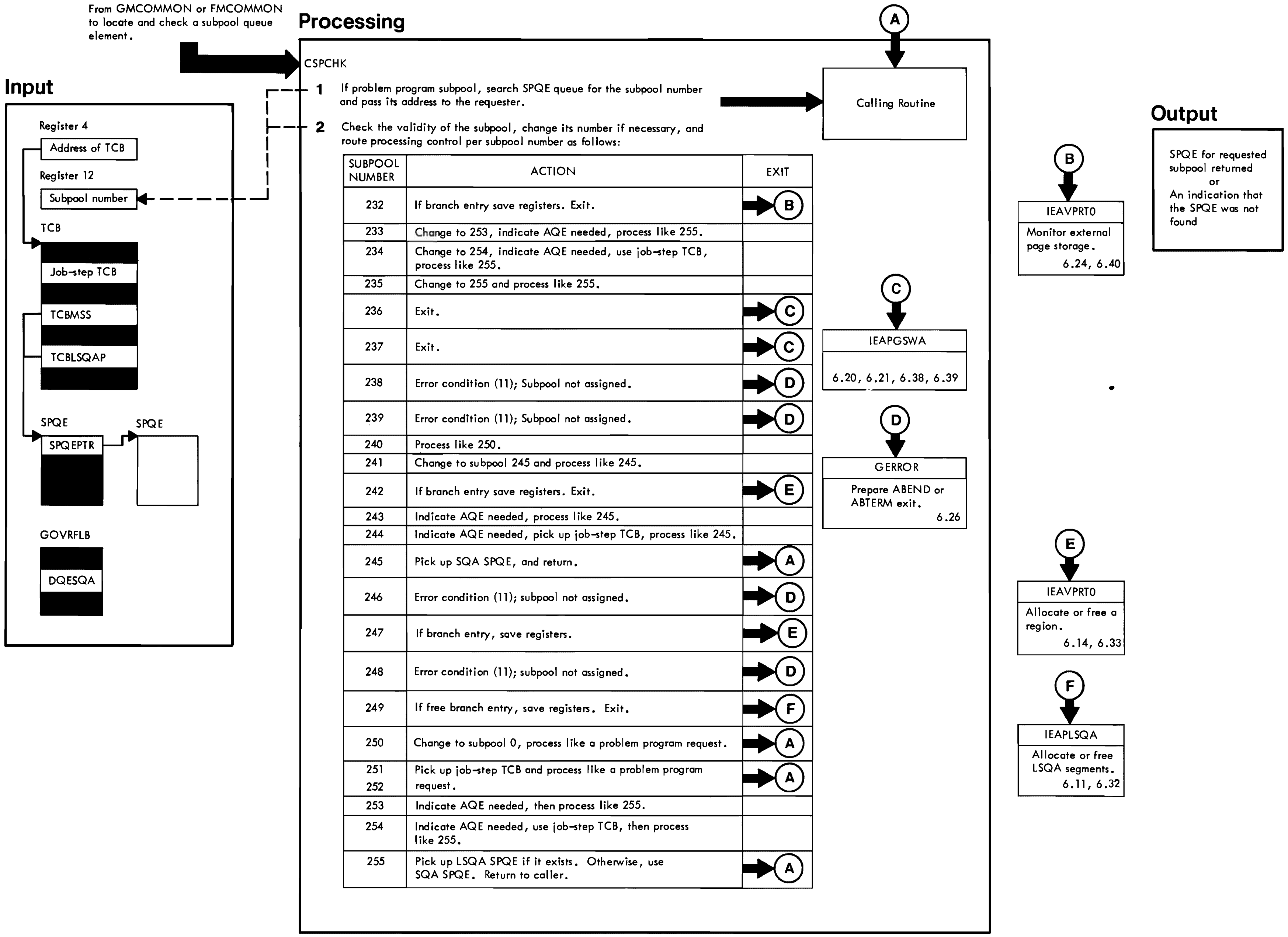
From GLIST when last enty has been processed.
From GNOTSAT 6.25, Step 6.

GCOMM7
GCOMM5
Return. → Caller

Output

Register 15
Return code=X'00'
Successful allocation

Diagram 6.3
Processing Subpools


From GMCOMMON or FMCOMMON to locate and check a subpool queue element.
Input
Register 4
Address of TCB
Register 12
Subpool number
TCB
Job-step TCB
TCBMSS
TCBLSQAP
SPQE
SPQEPTR
SPQE
GOVRFLB
DQESQA
Processing
CSPCHK
A
Calling Routine
B
IEAVPRTO
Monitor external page storage.
6.24, 6.40
C
IEAPGSWA
6.20, 6.21, 6.38, 6.39
D
GERROR
Prepare ABEND or ABTERM exit.
6.26
E
IEAVPRTO
Allocate or free a region.
6.14, 6.33
F
IEAPLSQA
Allocate or free LSQA segments.
6.11, 6.32
Output
SPQE for requested subpool returned or An indication that the SPQE was not found

CSPCHK

1 If problem program subpool, search SPQE queue for the subpool number and pass its address to the requester.

2 Check the validity of the subpool, change its number if necessary, and route processing control per subpool number as follows:

SUBPOOL NUMBER	ACTION	EXIT
232	If branch entry save registers. Exit.	B
233	Change to 253, indicate AQE needed, process like 255.	
234	Change to 254, indicate AQE needed, use job-step TCB, process like 255.	
235	Change to 255 and process like 255.	
236	Exit.	C
237	Exit.	C
238	Error condition (11); Subpool not assigned.	D
239	Error condition (11); Subpool not assigned.	D
240	Process like 250.	
241	Change to subpool 245 and process like 245.	
242	If branch entry save registers. Exit.	E
243	Indicate AQE needed, process like 245.	
244	Indicate AQE needed, pick up job-step TCB, process like 245.	
245	Pick up SQA SPQE, and return.	A
246	Error condition (11); subpool not assigned.	D
247	If branch entry, save registers.	E
248	Error condition (11); subpool not assigned.	D
249	If free branch entry, save registers. Exit.	F
250	Change to subpool 0, process like a problem program request.	A
251 252	Pick up job-step TCB and process like a problem program request.	A
253	Indicate AQE needed, then process like 255.	
254	Indicate AQE needed, use job-step TCB, then process like 255.	
255	Pick up LSQA SPQE if it exists. Otherwise, use SQA SPQE. Return to caller.	A

Diagram 6.3 Processing Subpools (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 If the requester is key 0 or in supervisor state, the subpool number is changed so that protected storage will be obtained.	CSPCHK	
2 Checks are made for two additional error conditions; a subpool number between 128 and 233, and a request for a subpool number greater than 233 for a nonsupervisor state program in key 0.		

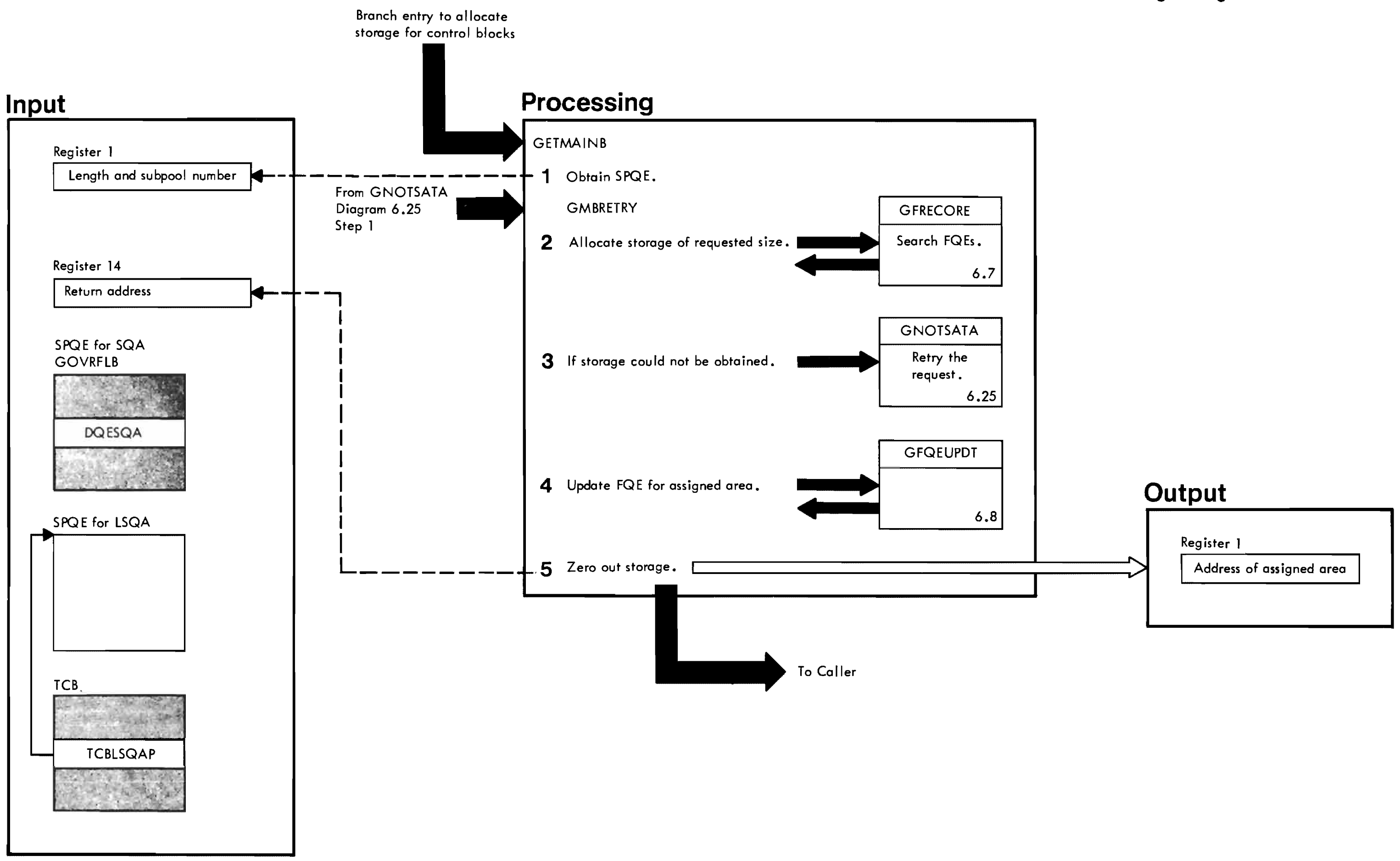
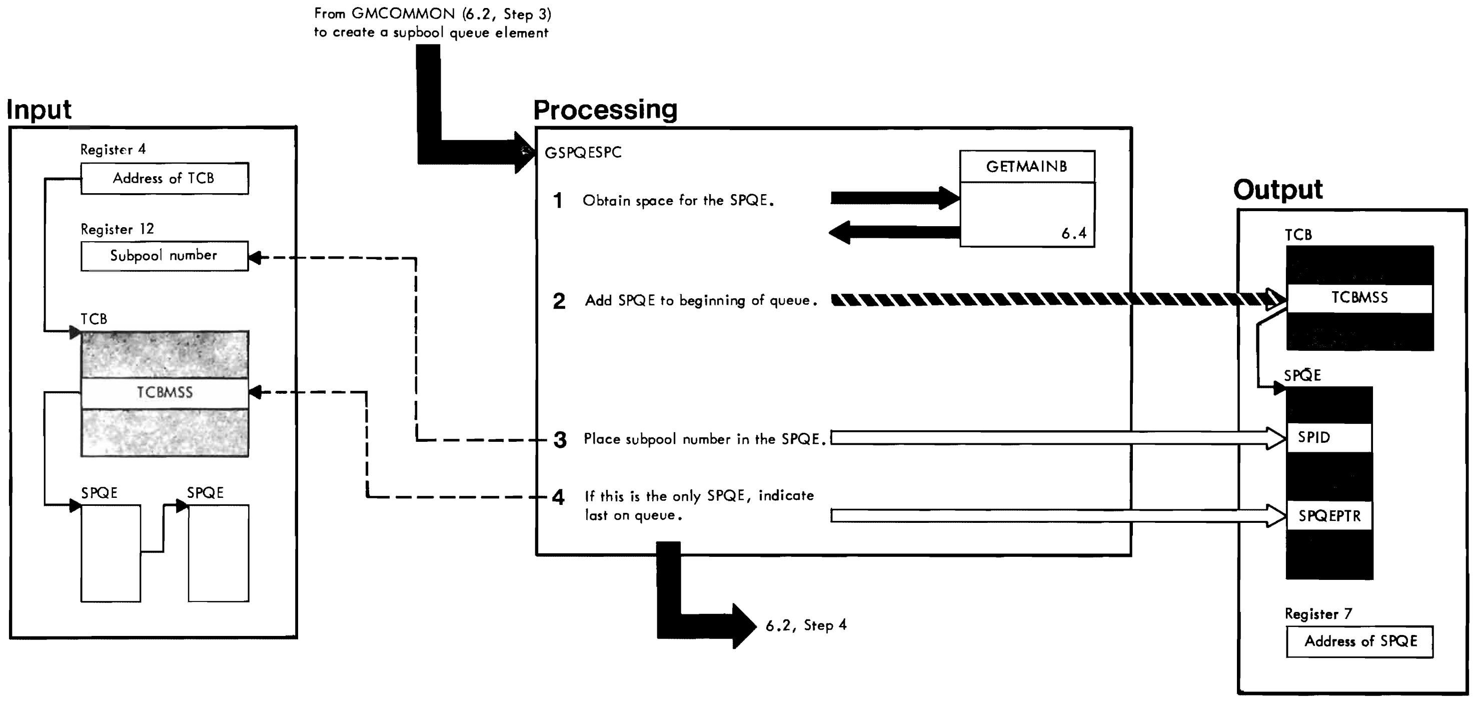

Branch entry to allocate storage for control blocks
Input
Register 1
Length and subpool number
Register 14
Return address
SPQE for SQA
GOVRFLB
DQESQA
SPQE for LSQA
TCB
TCBLSQAP
Processing
GETMAINB
1 Obtain SPQE.
From GNOTSATA Diagram 6.25 Step 1
GMBRETRY
2 Allocate storage of requested size.
GFRECORE
Search FQEs.
6.7
3 If storage could not be obtained.
GNOTSATA
Retry the request.
6.25
4 Update FQE for assigned area.
GFQEUPDT
6.8
5 Zero out storage.
To Caller
Output
Register 1
Address of assigned area

Diagram 6.4 Allocating Storage for Control Blocks (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
This is an internal subroutine used by the virtual storage supervision routines to obtain storage for control blocks in subpool 255 or 245.	GETMAINB	
2	GFRECORE	
3	GMBRETRY	
4	GPFEUPDT	

Diagram 6.5
Creating an SPQE

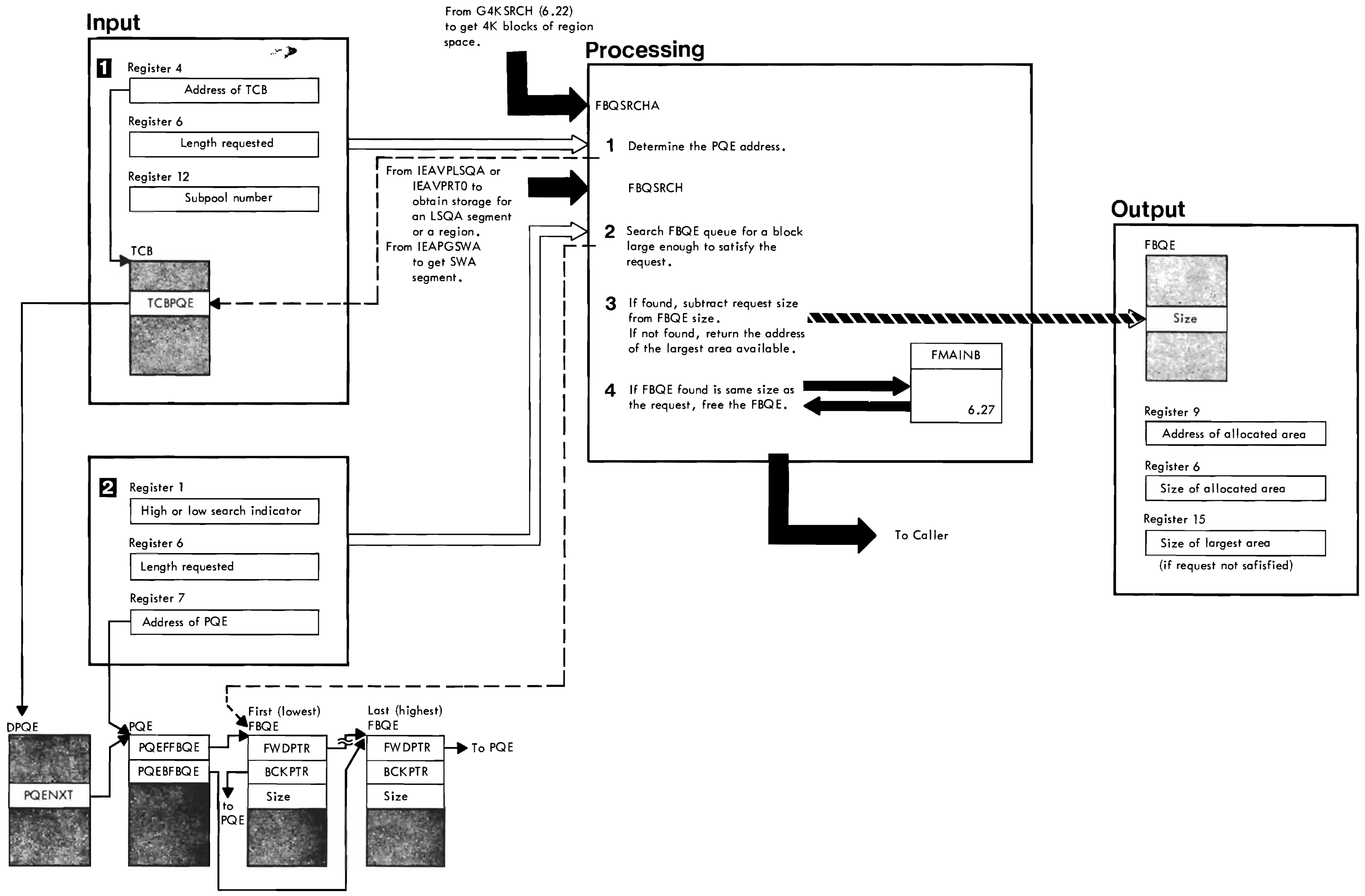


1 Obtain space for the SPQE.

2 Add SPQE to beginning of queue.

3 Place subpool number in the SPQE.

4 If this is the only SPQE, indicate last on queue.



1 Determine the PQE address.

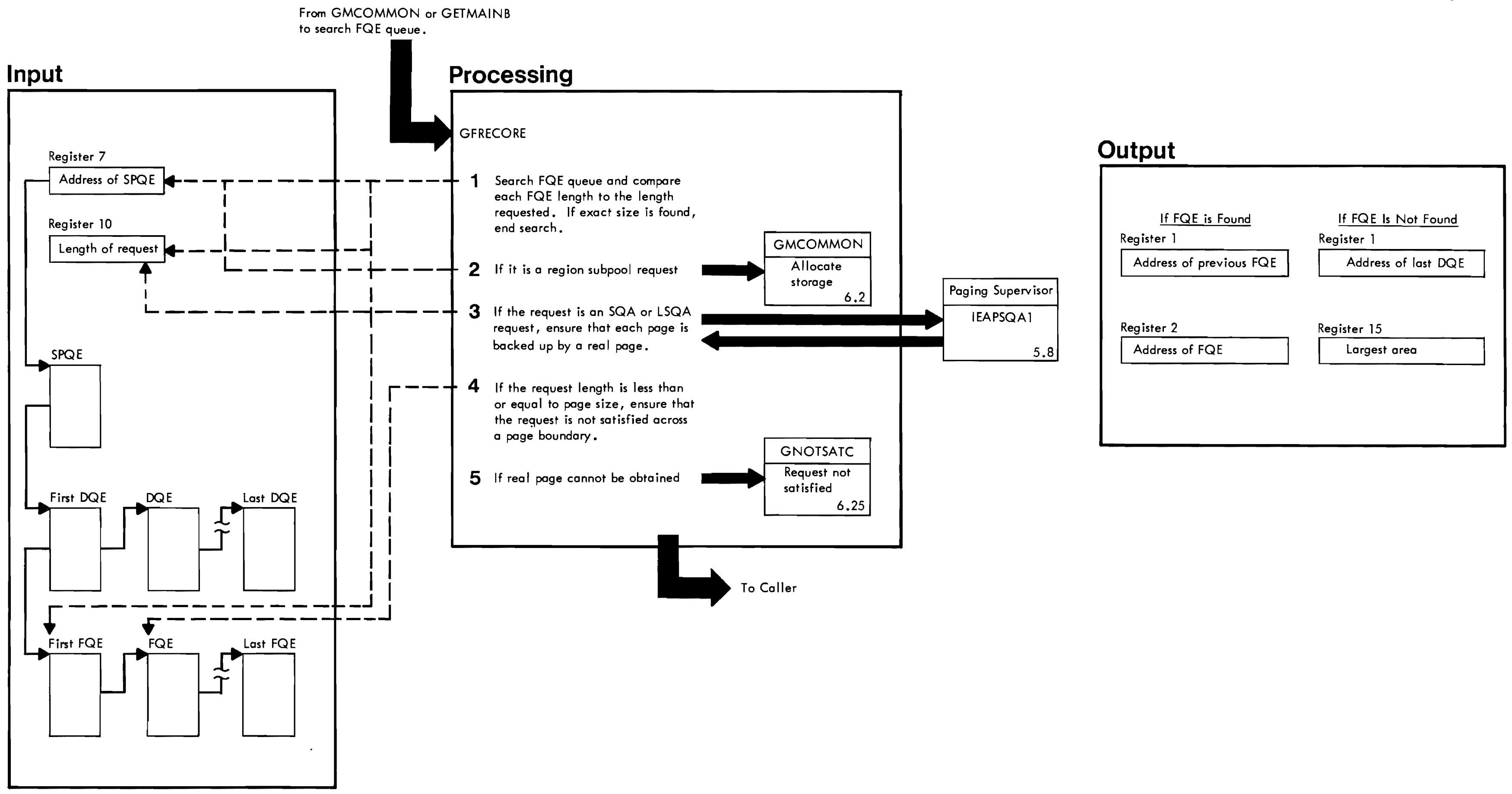
2 Search FBQE queue for a block large enough to satisfy the request.

3 If found, subtract request size from FBQE size. If not found, return the address of the largest area available.

4 If FBQE found is same size as the request, free the FBQE.

Diagram 6.6 Searching FBQEs (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 When entered at FBQSRCHA from G4KSRCH to obtain 4K multiples of region space, the search of the FBQEs is made through the backward pointers.	FBQSRCHA	
2 When entered at FBQSRCH, register 1 indicates the direction of the FBQE search. A value of 4 indicates the request is to be satisfied by starting the search with the high addresses. A value of 0 indicates the low addresses are to be searched first.	FBQSRCH	



1 Search FQE queue and compare each FQE length to the length requested. If exact size is found, end search.

2 If it is a region subpool request

3 If the request is an SQA or LSQA request, ensure that each page is backed up by a real page.

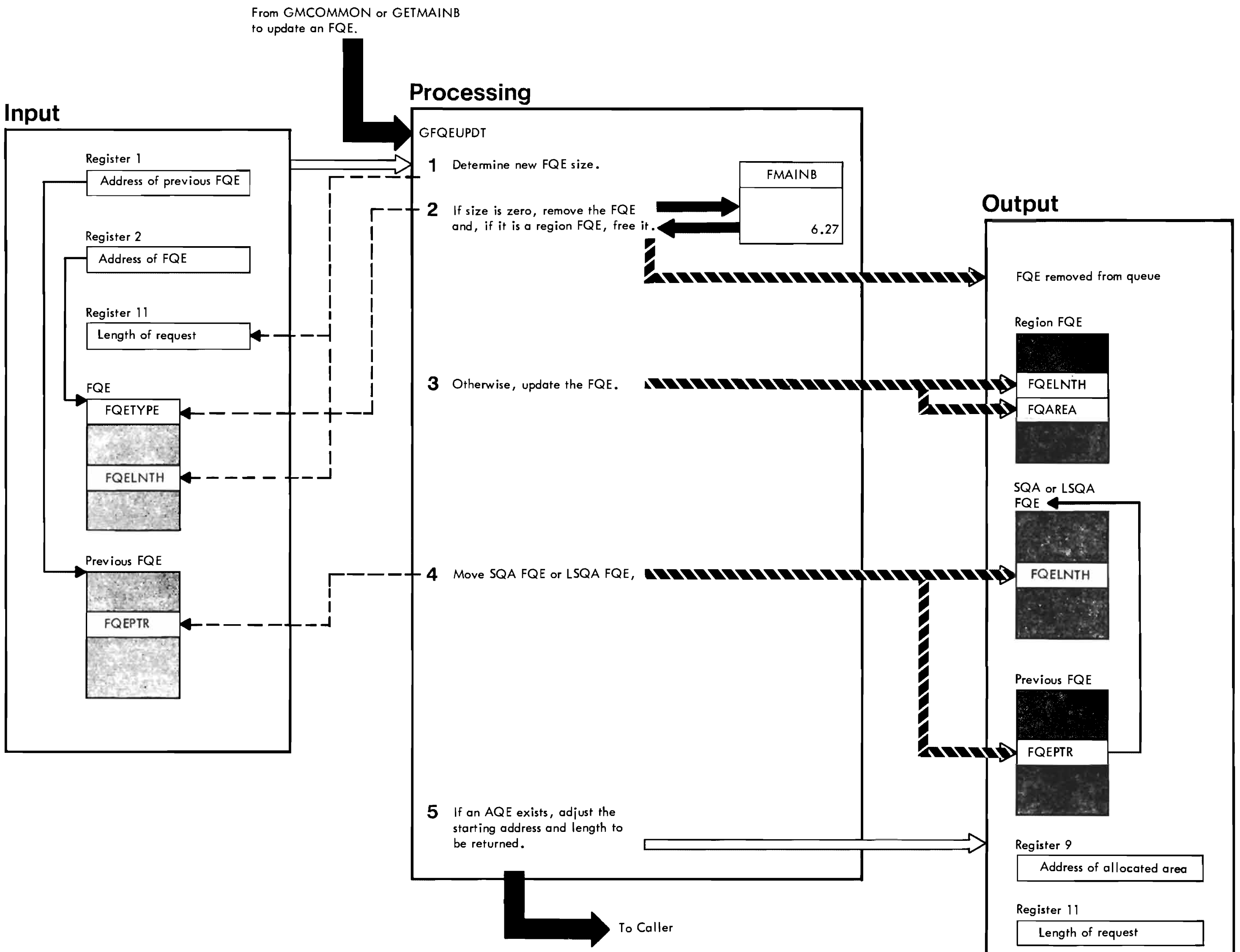
4 If the request length is less than or equal to page size, ensure that the request is not satisfied across a page boundary.

5 If real page cannot be obtained

Diagram 6.7 Searching FQEs (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 The GFRECORE routine searches the FQE queues for the requested subpool to find the FQE that best fits the length requested.	GFRECORE	
2 For region subpools, the FQE of the requested size or the largest available FQE (in the case of a variable request) is passed back to the requester (GMCOMMON) along with a pointer to the last DQE.		
3 A series of calls is made to the paging supervisor to obtain real pages to back up each page that will be allocated to satisfy the LSQA or SQA request.	IEAPSQA	
4 LSQA and SQA requests for 4K or less are not satisfied across page boundaries.	GFRECORE	
5 If real pages cannot be obtained for all the requested LSQA or SQA pages, those pages that were obtained are cleared and the request is not satisfied.	TRNONPVT IEAPSIER	

Diagram 6.8
Updating FQEs to Allocate Space



GFQEUPDT

1 Determine new FQE size.

2 If size is zero, remove the FQE and, if it is a region FQE, free it.

3 Otherwise, update the FQE.

4 Move SQA FQE or LSQA FQE,

5 If an AQE exists, adjust the starting address and length to be returned.

Diagram 6.8 Updating FQEs to Allocate Space (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 The new FQE size is equal to the original FQE size minus the length of the request.	GFQEUPDT	
2 If the FQETYPE field in the FQE is equal to X'80', the FQE is freed.	FMAINB	
3 If the FQE is for a region, only the area and length fields must be modified.	GFQEUPDT	
4 If the FQE is for an LSQA or SQA, it must be moved to the new free area at the original location minus the requested length, and the previous FQE must be updated to point to the new location.		
5 If an AQE was built, eight is added to the starting address of the allocated area and eight is subtracted from the length returned to the caller.		

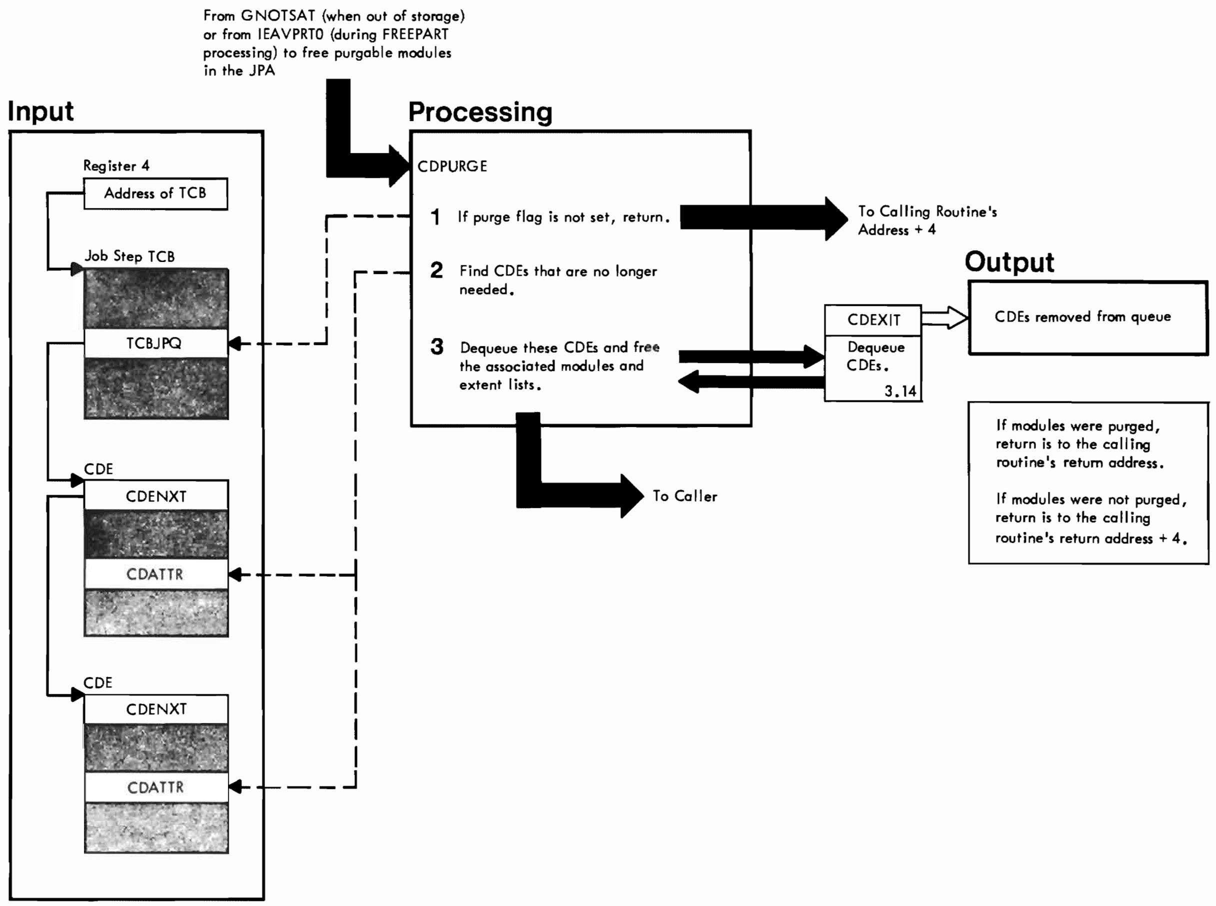
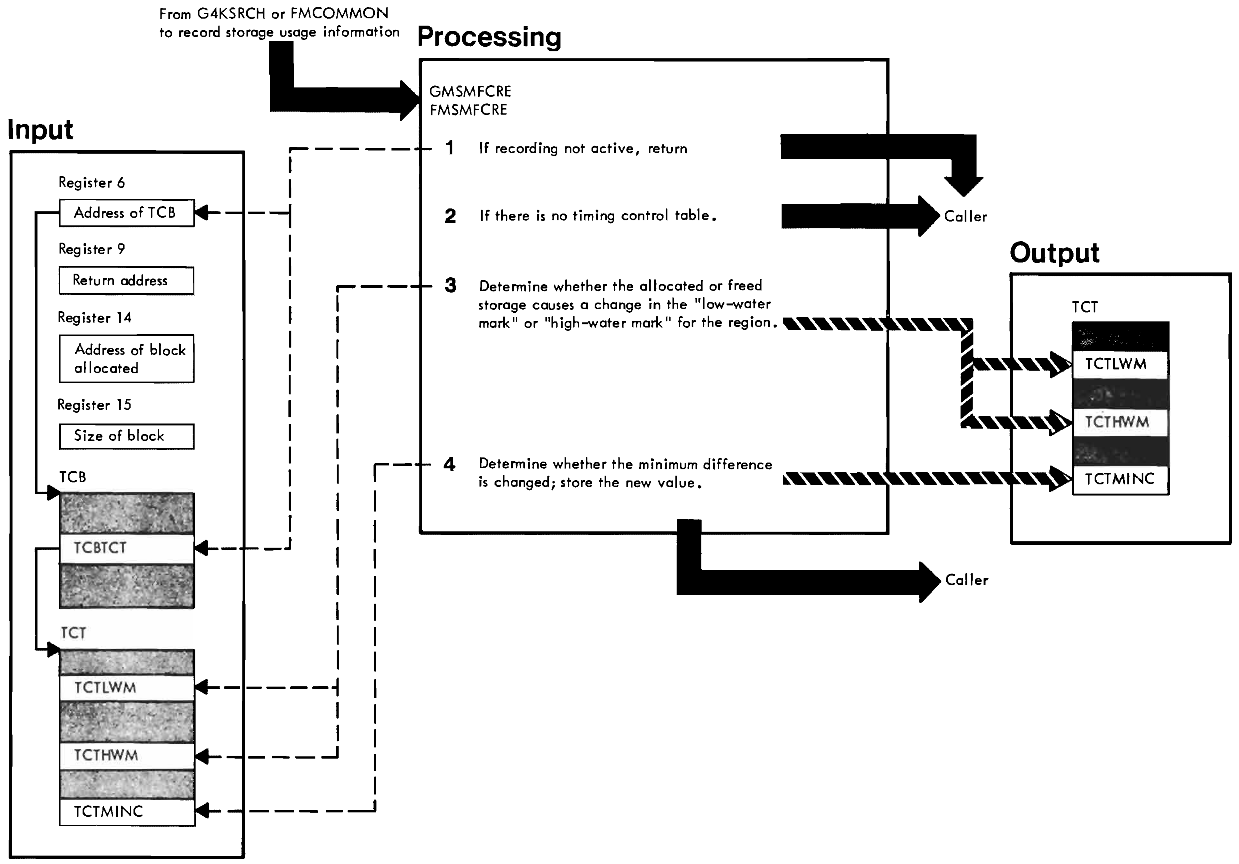

From GNOTSAT (when out of storage) or from IEAVPRT0 (during FREEPART processing) to free purgable modules in the JPA
Input
Register 4
Address of TCB
Job Step TCB
TCBJPQ
CDE
CDENXT
CDATTR
CDE
CDENXT
CDATTR
Processing
CDPURGE
1 If purge flag is not set, return.
2 Find CDEs that are no longer needed.
3 Dequeue these CDEs and free the associated modules and extent lists.
To Calling Routine's Address + 4
To Caller
CDEXIT
Dequeue CDEs.
3.14
Output
CDEs removed from queue
If modules were purged, return is to the calling routine's return address.
If modules were not purged, return is to the calling routine's return address + 4.

Diagram 6.9 Purging Modules (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 The purge flag is in the TCBJPQ field of the TCB. When set to X'80' it indicates that the modules can be purged.	CDPURGE GNOTSAT IEAVPRT0	
2 Each CDE that has the "release" flag set in its attributes field represents a module that no longer is needed in the region. That is, there are no outstanding requests for the module by any routine in the job step.	CDPURGE	
3 For each module that is no longer needed, the CDPURGE routine branches to CDDESTRY in CDEXIT to dequeue the CDE and free the associated module and extent list. If return is made to GNOTSAT with modules purged, the request is retried.	CDEXIT GNOTSAT IEAVPRT0	CDDESTRY

Diagram 6.10
Updating the TCT



1 If recording not active, return

2 If there is no timing control table.

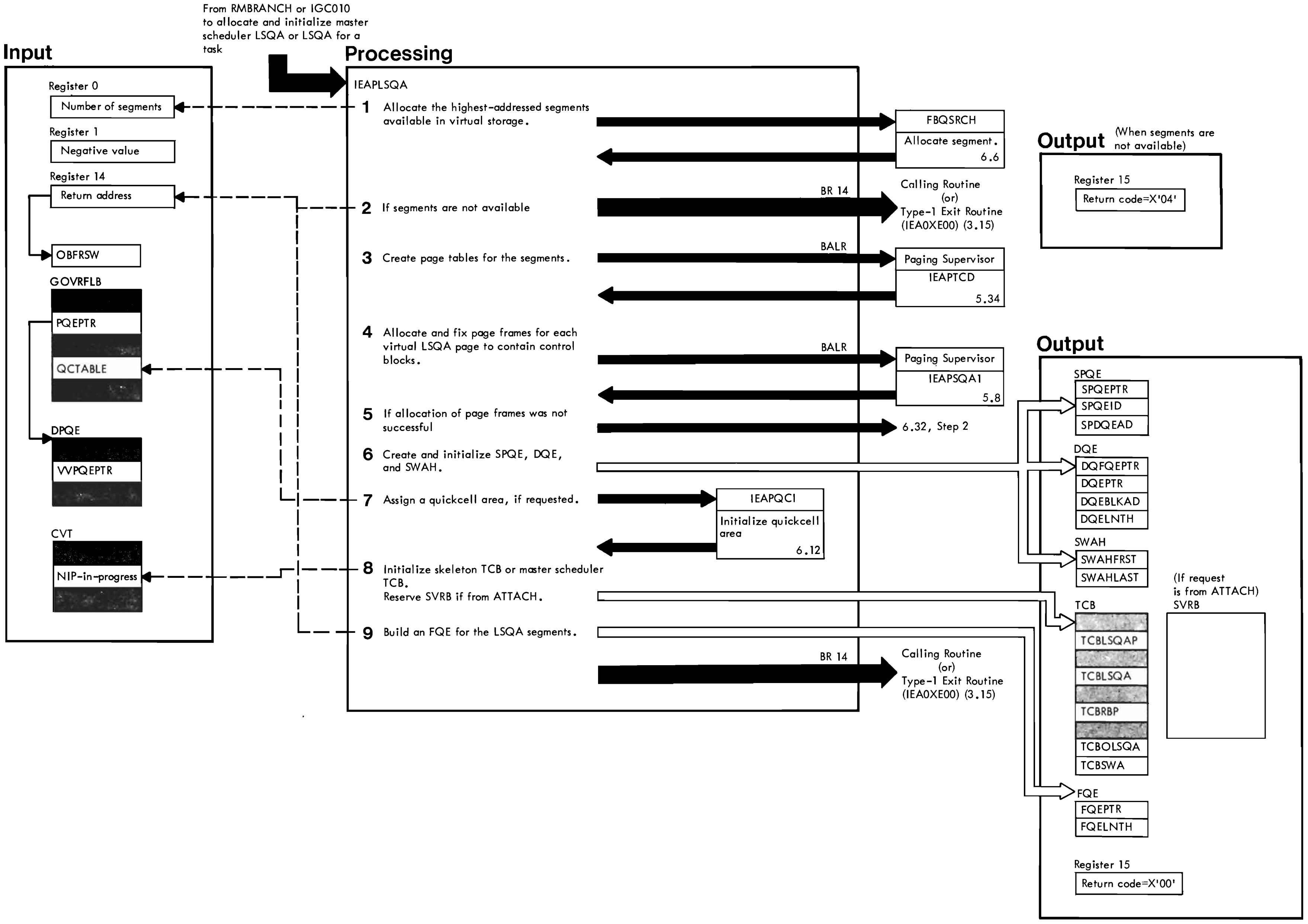
3 Determine whether the allocated or freed storage causes a change in the "low-water mark" or "high-water mark" for the region.

4 Determine whether the minimum difference is changed; store the new value.

Diagram 6.10 Updating the TCT (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
2 If there is no TCT, SMF storage information is not being recorded for this program.	GMSMFCRE FMSMFCRE	
3 The "low-water mark" is the address of the highest storage address allocated from the bottom of the region. The "high-water mark" is the address of the lowest storage address allocated from the top of the region. If either is changed, the new value is recorded in the TCT.		
4 The difference is calculated in multiples of 2K.	GMSMFCRE	

Diagram 6.11
Allocating LSQA Segments



1 Allocate the highest-addressed segments available in virtual storage.

2 If segments are not available

3 Create page tables for the segments.

4 Allocate and fix page frames for each virtual LSQA page to contain control blocks.

5 If allocation of page frames was not successful

6 Create and initialize SPQE, DQE, and SWAH.

7 Assign a quickcell area, if requested.

8 Initialize skeleton TCB or master scheduler TCB. Reserve SVRB if from ATTACH.

9 Build an FQE for the LSQA segments.

Diagram 6.11 Allocating LSQA Segments (Module IEAPLSQA)

NOTES	ROUTINE NAME	LABEL
1	IEAPLSQA FBQSRCH	GETLSQA
2 Control is returned to the Type-1 Exit routine if the branch entry switch (OBFRSW) is off.		
3 The Create/Destroy Page Table routine of the paging supervisor is used to create the page tables and to initialize segment table entries for the allocated segments.	IEAPTCD	
4 The SQA/LSQA allocation subroutine of the paging supervisor fixes page frames in real storage to back up each allocated virtual LSQA page.	IEAPLSQA IEAPSQA1	INITLSQA
6 The control blocks are initialized as follows:		

Control Block	Field Name	Dec. Disp.	Bytes	Initialized to
SPQE	SPQEPTR	1	3	Zero
	SPQEID	4	1	X'FF'
	SPDQEPTR	5	1	LSQA DQE address
DQE	DQFQEPTR	0	4	LSQA FQE address for LSQA
	DQEPTR*	4	4	QCDBLK address
	DQEBLKAD	8	4	LSQA address for first segment
	DQELNTH	12	4	Size of LSQA
SWAH	SWAHFRST	0	4	Zero
	SWAHLAST	4	4	Zero

*When initializing the DQEPTR field, the routine first checks the value in QCTABLE (in GOVRFLB). If it is zero (indicating that no LSQA quickcells have been requested), the routine sets the DQEPTR field to zero. Otherwise, the routine calculates the address at which the QCDBLK is to be built and stores it in DQEPTR.

8 No TCB is assigned if the request is a master scheduler LSQA request; the fields are simply filled in. If the request is not for the master scheduler, an SVRB is reserved.

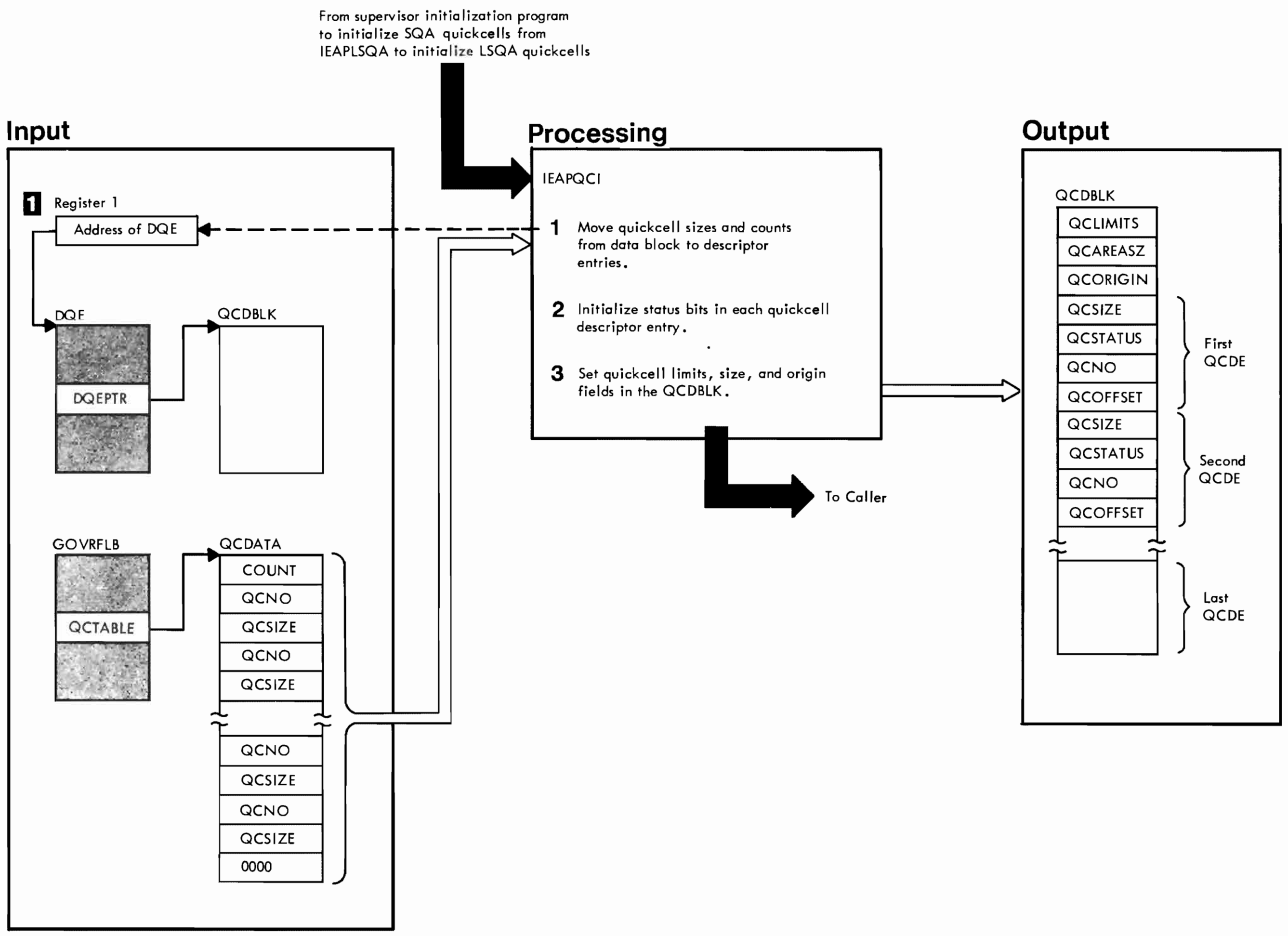
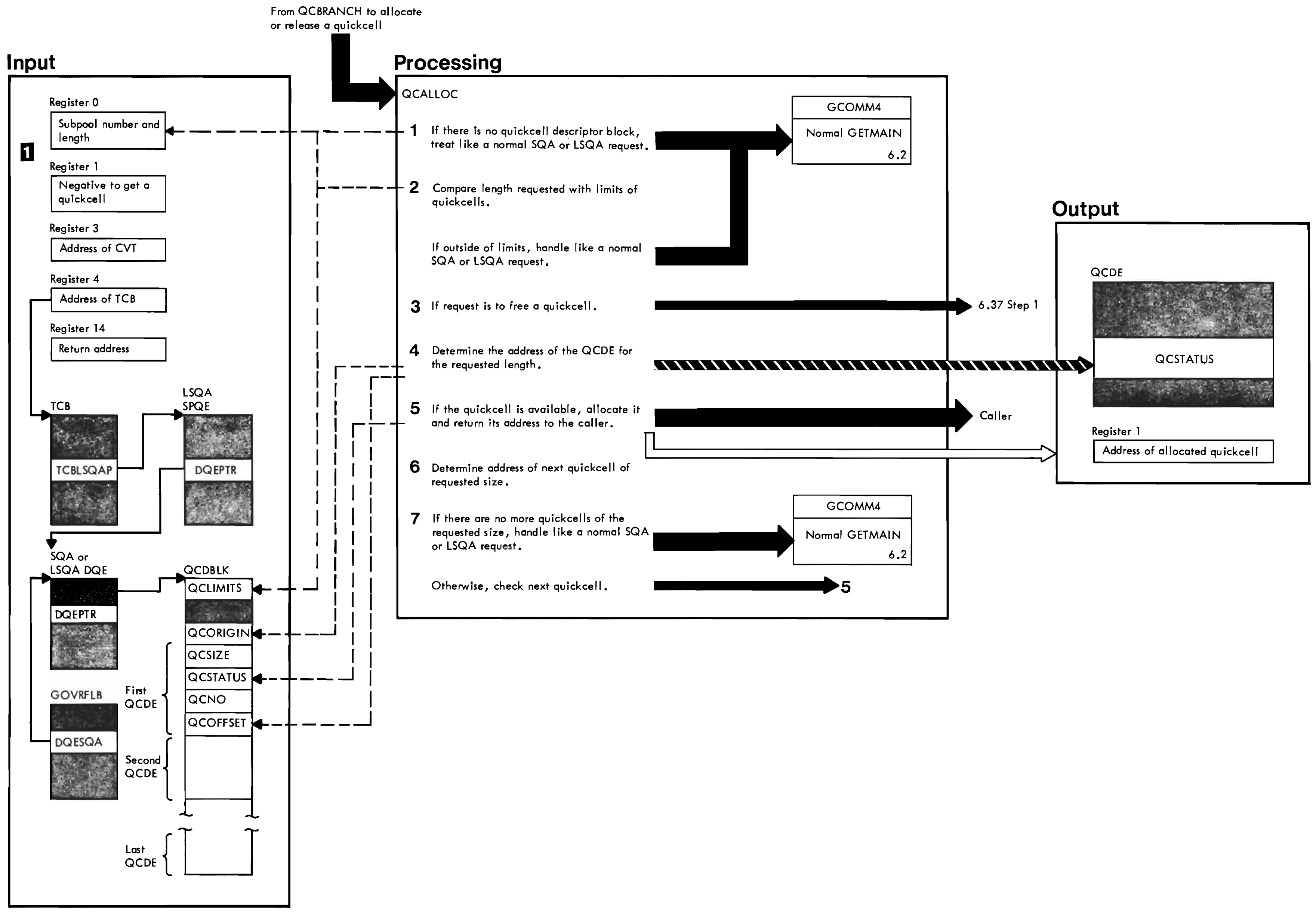

From supervisor initialization program to initialize SQA quickcells from IEAPLSQA to initialize LSQA quickcells
Input
1 Register 1
Address of DQE
DQE
DQEPTR
QCDBLK
GOVRFLB
QCTABLE
QCDATA
COUNT
QCNO
QCSIZE
QCNO
QCSIZE
QCNO
QCSIZE
QCNO
QCSIZE
0000
Processing
IEAPQCI
1 Move quickcell sizes and counts from data block to descriptor entries.
2 Initialize status bits in each quickcell descriptor entry.
3 Set quickcell limits, size, and origin fields in the QCDBLK.
To Caller
Output
QCDBLK
QCLIMITS
QCAREASZ
QCORIGIN
QCSIZE
QCSTATUS
QCNO
QCOFFSET
First QCDE
QCSIZE
QCSTATUS
QCNO
QCOFFSET
Second QCDE
Last QCDE

Diagram 6.12 Initializing LSQA Quickcells Areas (Module IEAPLSQA)

NOTES	ROUTINE NAME	LABEL
1 The first word of the quickcell data area, pointed to by the QCTABLE field of GOVRFLB, contains the count of the bytes needed for the quickcell area. Following this, the data area consists of a series of halfword entries, each describing a quickcell area and the number of cells reserved for that size, with the last entry being a dummy having zero in the first byte. The caller must place the address at which the quickcell descripter block (QCDBLK) is to be built in the DQEPTR field of the DQE and pass the address of the DQE in register 1.	IEAPQCI	



QCALLOC

1 If there is no quickcell descriptor block, treat like a normal SQA or LSQA request.

2 Compare length requested with limits of quickcells.

If outside of limits, handle like a normal SQA or LSQA request.

3 If request is to free a quickcell.

4 Determine the address of the QCDE for the requested length.

5 If the quickcell is available, allocate it and return its address to the caller.

6 Determine address of next quickcell of requested size.

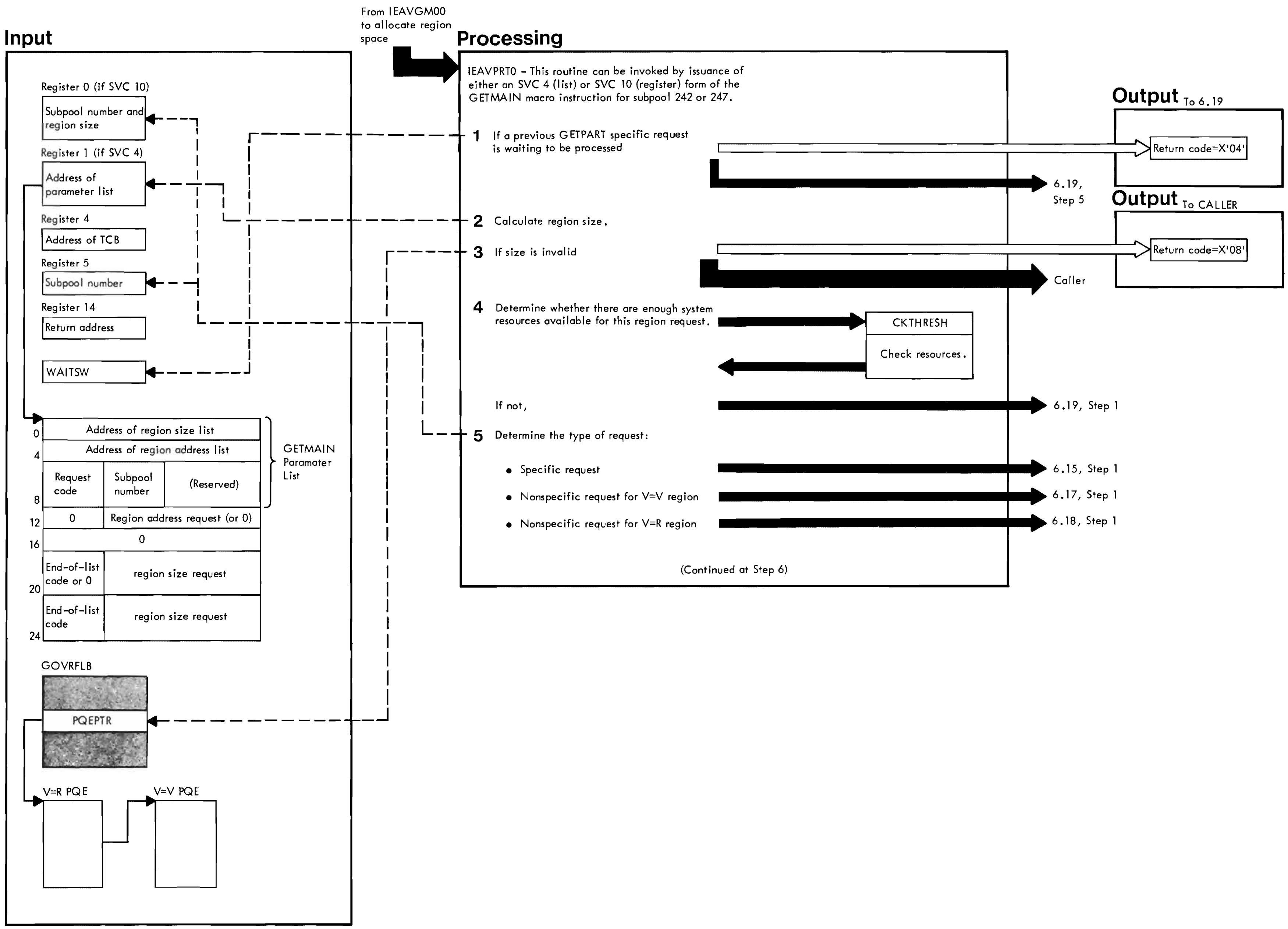
7 If there are no more quickcells of the requested size, handle like a normal SQA or LSQA request.

Otherwise, check next quickcell.

Diagram 6.13 Allocating a Quickcell (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 The SQA and LSQA quickcell areas are each described by a quickcell descriptor block (QCDBLK). The QCDBLK for SQA quickcells is built in the SQA during supervisor initialization. The QCDBLK for LSQA quickcells is built when the LSQA is allocated (see Diagram 6.12).		
1 The routine obtains the address of the QCDBLK from the DQEPTR field of the LSQA or SQA DQE for subpools 255 and 245 respectively. If the task has no LSQA, subpool 255 requests are diverted to the SQA by changing the subpool number to 245 and the QCDBLK address to zero. Such requests are treated like normal SQA requests not like quickcell requests.	QCALLOC	
2 The length passed in register 1 is compared to the QCLIMITS field which indicates the largest quickcell length in the set.		
3 The address of the QCDE for the requested length is determined as follows: Address of QCDE = Address of first QCDE + (length-8) /2		
4 The QCSTATUS field indicates whether the quickcell is available for allocation. The address of the first quickcell is determined as follows: First quickcell address = QCORIGIN+QCOFFSET		
5 The address of the next quickcell for the requested length is calculated by adding the request length to the previous quickcell address.		
6 Each time a quickcell of the specified length is checked for availability, a counter is decremented which, when zero, indicates they have all been checked.		

• Diagram 6.14 (Steps 1-5) Allocating a Region



IEAVPRT0 - This routine can be invoked by issuance of either an SVC 4 (list) or SVC 10 (register) form of the GETMAIN macro instruction for subpool 242 or 247.

1 If a previous GETPART specific request is waiting to be processed

2 Calculate region size.

3 If size is invalid

4 Determine whether there are enough system resources available for this region request.

If not,

5 Determine the type of request:

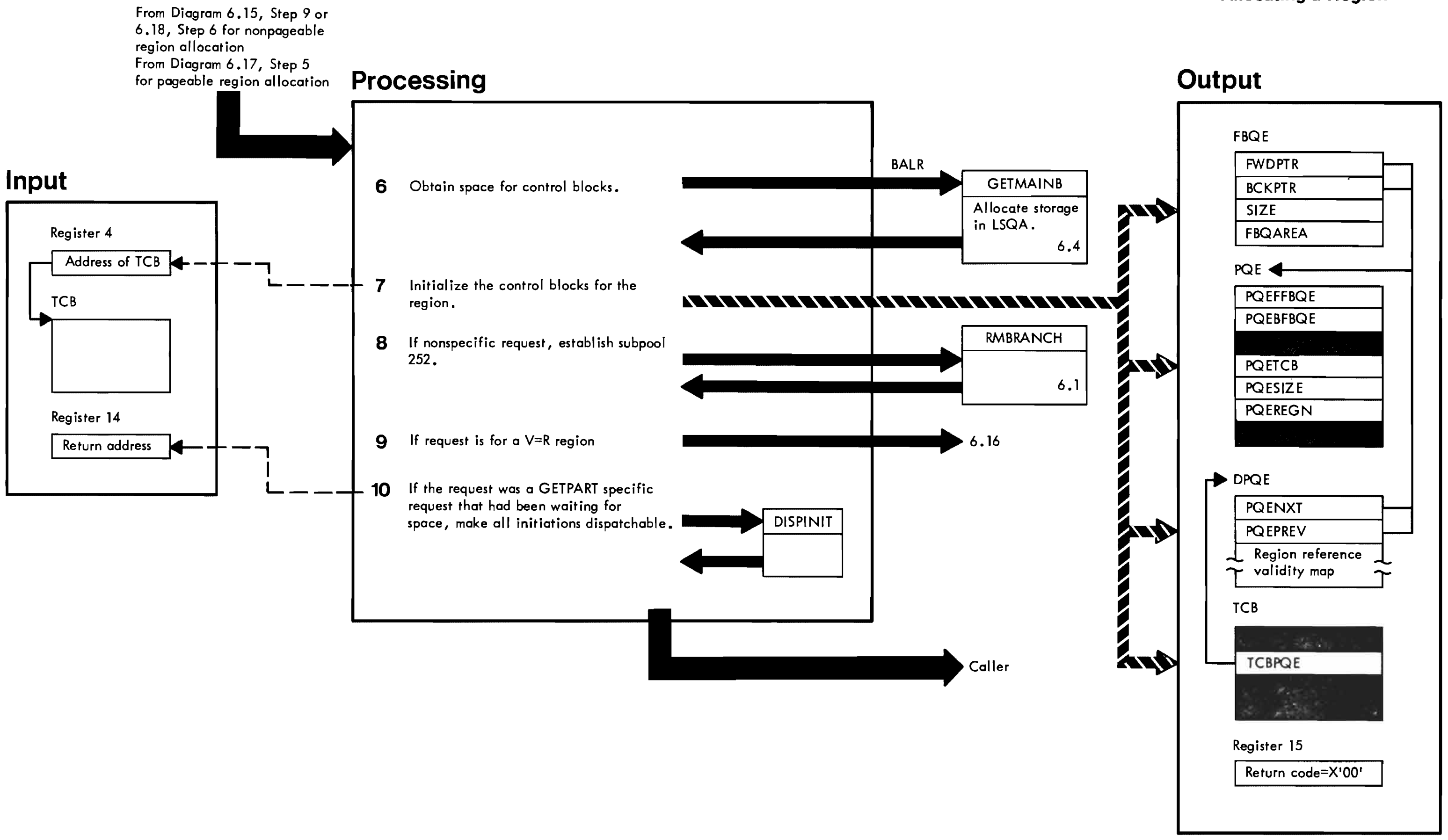
- Specific request
- Nonspecific request for V=V region
- Nonspecific request for V=R region

(Continued at Step 6)

Diagram 6.14 (Steps 1-5) Allocating a Region (Module IEAVPRT0)

NOTES	ROUTINE NAME	LABEL
1 If WAITSW has a value of X'0F' and the address of the requesting TCB is not the same as the TCB waiting for region space (TCBSAVE), this request will not be processed immediately.	IEAVPRT0	
2 The total length of the request is calculated as follows: _Request_ — _Total Length_ Single region = — Requested length Split region nonspecific = — Sum of lengths Split region specific = — Invalid; code X'08'		
3 The requested size is rounded upward to a segment multiple for nonpageable region requests, or to a page multiple for pageable region requests. Then the adjusted size is compared to the PQE size (V=R or V=V). If the requested size is larger, control is returned to the calling routine.		
4 The CKTHRESH subroutine branches to the paging supervisor at entry point IEAPCHTH, passing to it the size of the requested region. If the paging supervisor has insufficient resources (thrashing), CKTHRESH returns with a return code of 4 in register 15. Otherwise, CKTHRESH returns with a code of zero in register 15.	CKTHRESH IEAPCHTH	
5 Specific requests support Checkpoint/Restart.		

Diagram 6.14 (Steps 6-10) Allocating a Region



6 Obtain space for control blocks.

7 Initialize the control blocks for the region.

8 If nonspecific request, establish subpool 252.

9 If request is for a V=R region

10 If the request was a GETPART specific request that had been waiting for space, make all initiations dispatchable.

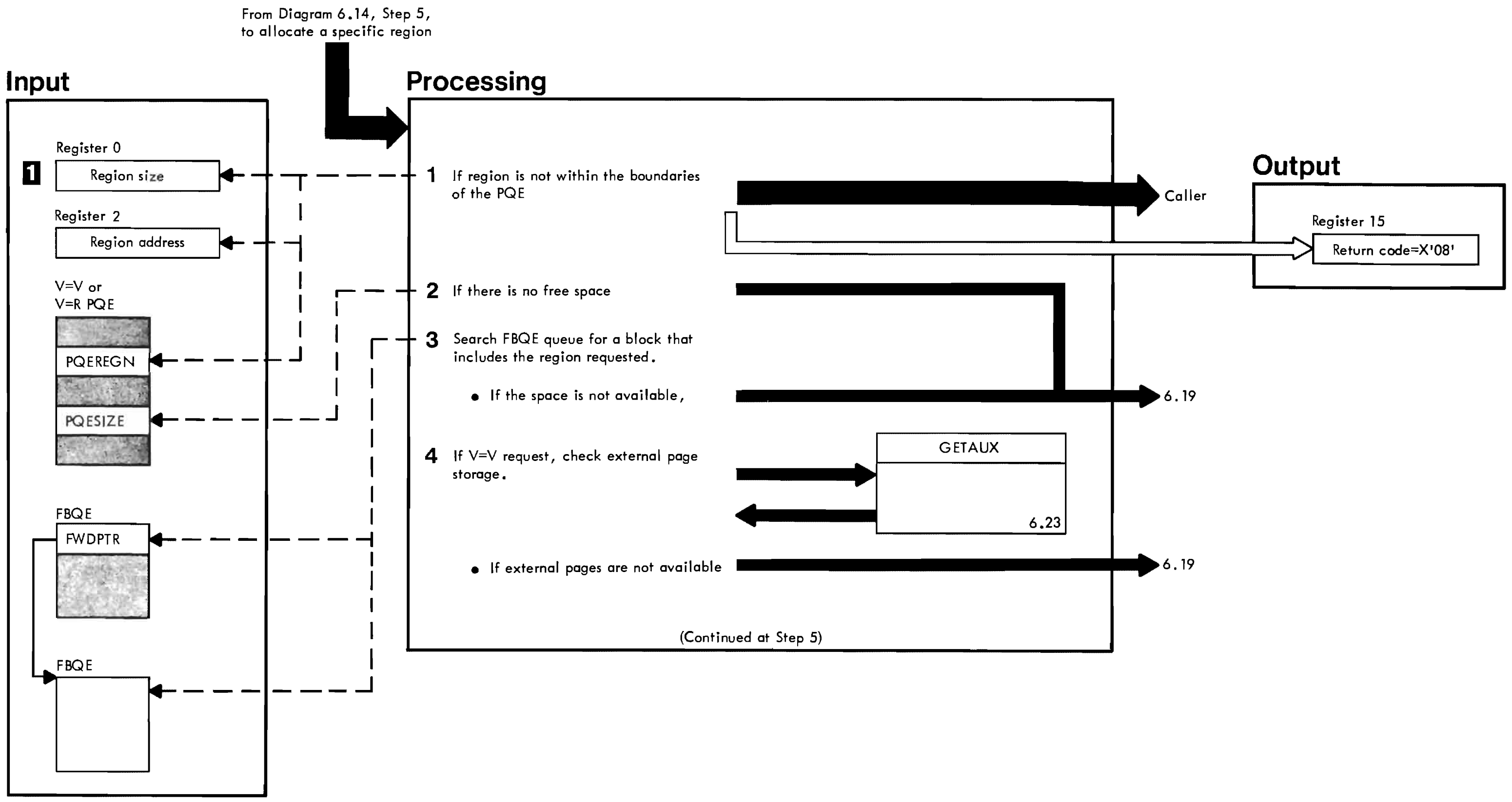
Diagram 6.14 (Steps 6-10) Allocating a Region (Module IEAVPRT0)

NOTES	ROUTINE NAME	LABEL
6 GETPART branches to the GETMAINB subroutine three times to obtain (1) 16 bytes for an FBQE (2) 48 bytes for a DPQE and region reference-validity map, and (3) 32 bytes for a PQE.	GETMAINB	
7 GETPART stores the address of the DPQE - 8 in the TCB and then initializes other control block fields as follows:	GETPART	

Control Block	Field Name	Dec. Disp.	Hex. Disp.	Bytes	Initialized to
FBQE	FWDPTR	0	0	4	PQE address
	BCKPTR	4	4	4	PQE address
	SIZE	8	8	4	Region size
	REGADDR	12	C	4	Region start address
PQE	PQEFFBQE	0	0	4	FBQE address
	PQELFBQE	4	4	4	FBQE address
	PQETCB	16	10	4	TCB address
	PQESIZE	20	14	4	Region size
	PQEREGN	24	18	4	Region start address
DPQE	PQENXT	0	0	4	PQE address
	PQEPREV	4	4	4	PQE address

Region reference-validity map

Diagram 6.15 (Steps 1-4) Allocating a Specific Region



1 If region is not within the boundaries of the PQE

2 If there is no free space

3 Search FBQE queue for a block that includes the region requested.

- If the space is not available,

4 If V=V request, check external page storage.

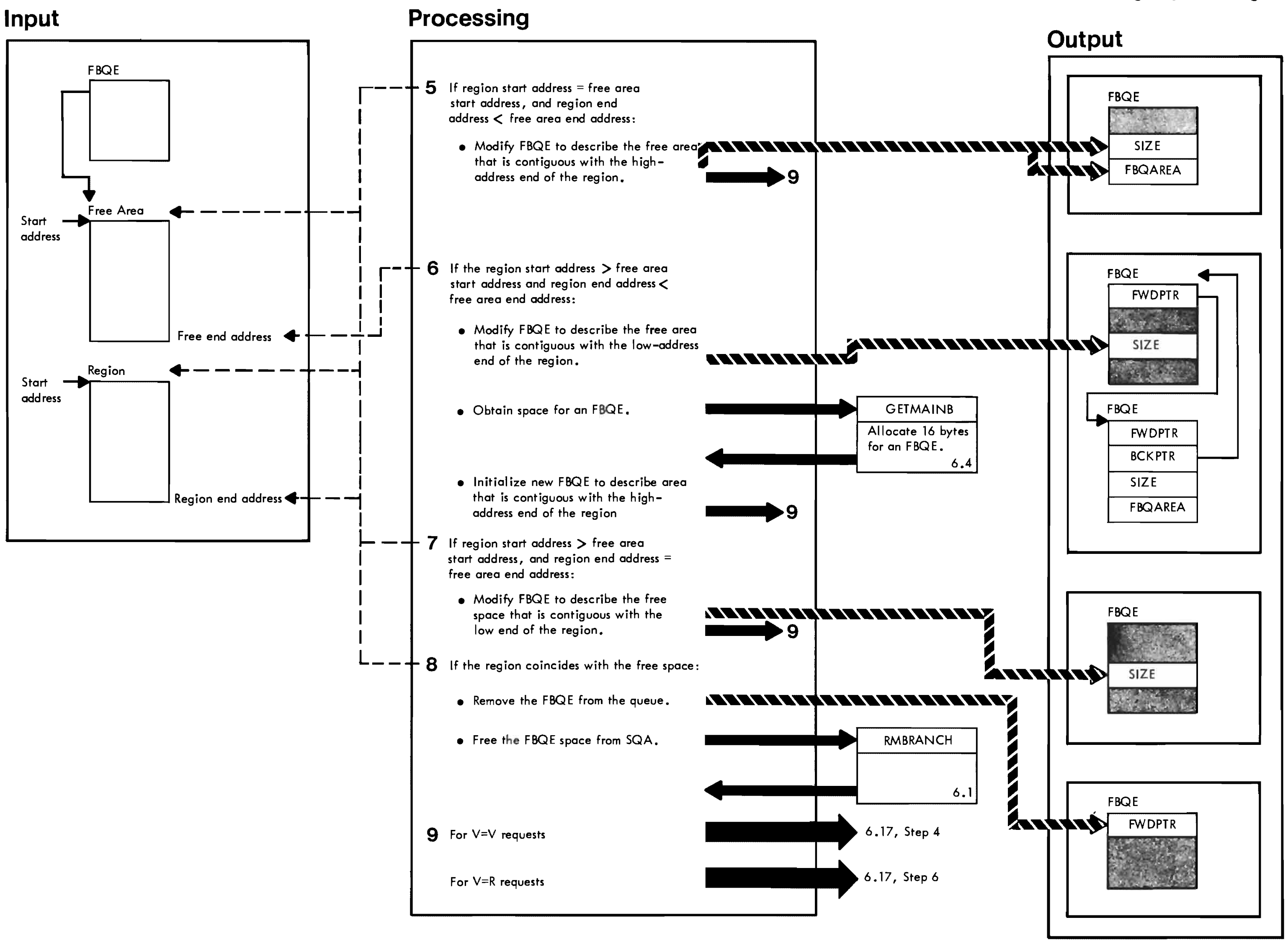
- If external pages are not available

(Continued at Step 5)

Diagram 6.15 (Steps 1-4) Allocating a Specific Region (Module IEAVPRTO)

NOTES	ROUTINE NAME	LABEL
1 The address of the V=V or V=R PQE, the rounded region size, and the specific address of the region are determined prior to entering the GETPART specific code.		
1 If any of the following conditions exist, the region is not within the boundaries of the dynamic area: region start address < dynamic area start address region start address > dynamic area end address region end address > dynamic area end address	GETPART	

Diagram 6.15 (Steps 5-9)
Allocating a Specific Region



5 If region start address = free area start address, and region end address < free area end address:

- Modify FBQE to describe the free area that is contiguous with the high-address end of the region. → 9

6 If the region start address > free area start address and region end address < free area end address:

- Modify FBQE to describe the free area that is contiguous with the low-address end of the region.
- Obtain space for an FBQE.
- Initialize new FBQE to describe area that is contiguous with the high-address end of the region → 9

7 If region start address > free area start address, and region end address = free area end address:

- Modify FBQE to describe the free space that is contiguous with the low end of the region. → 9

8 If the region coincides with the free space:

- Remove the FBQE from the queue.
- Free the FBQE space from SQA.

9 For V=V requests

For V=R requests

Diagram 6.16
Allocating a Specific V=R Region

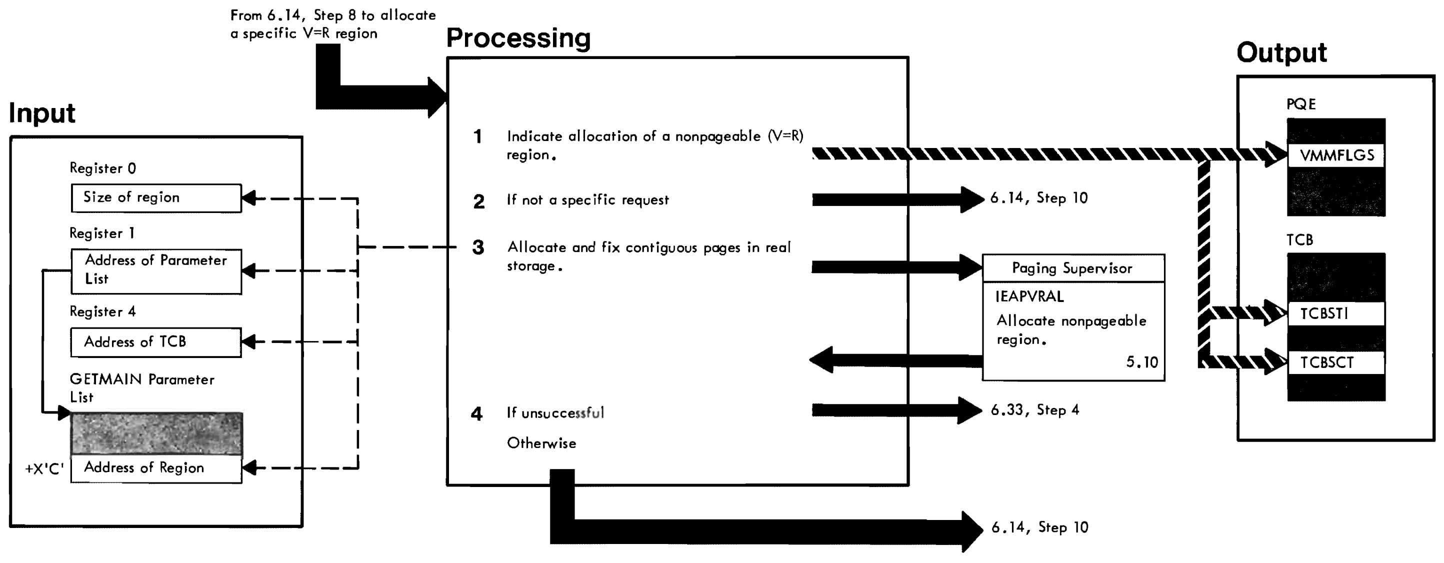


Diagram 6.17
Allocating a Nonspecific V=V Region

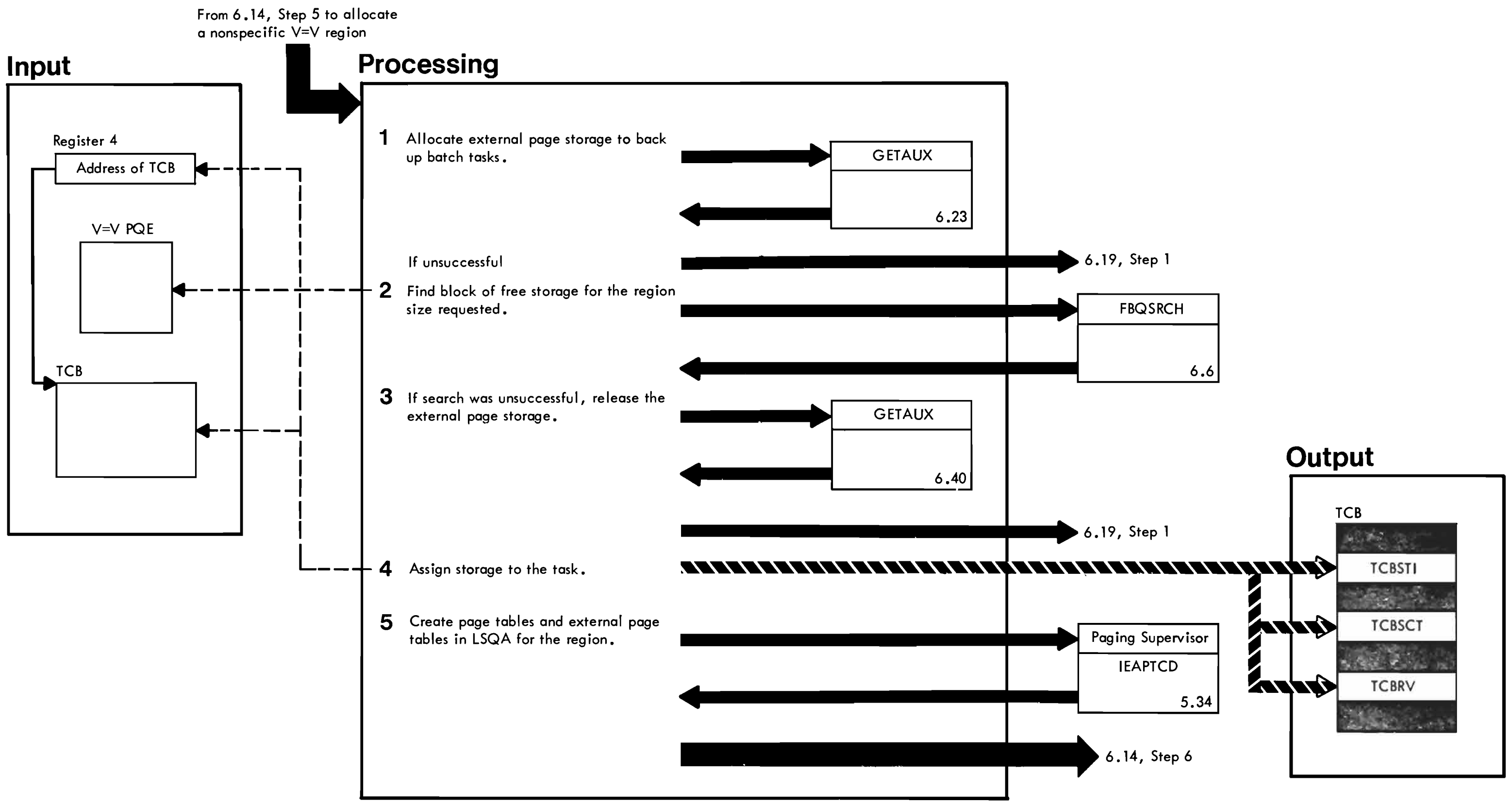
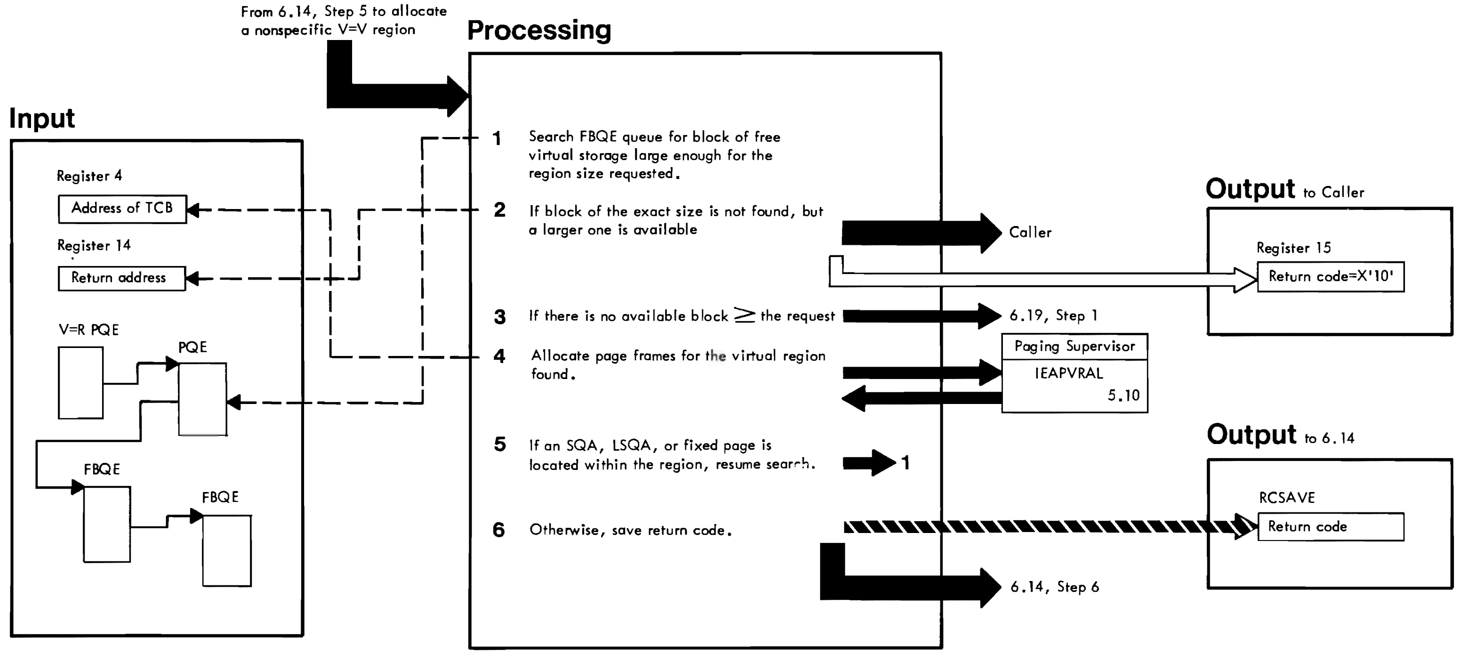

From 6.14, Step 5 to allocate a nonspecific V=V region
Input
Register 4
Address of TCB
V=V PQE
TCB
Processing
1 Allocate external page storage to back up batch tasks.
GETAUX
6.23
If unsuccessful
6.19, Step 1
2 Find block of free storage for the region size requested.
FBQSRCH
6.6
3 If search was unsuccessful, release the external page storage.
GETAUX
6.40
6.19, Step 1
4 Assign storage to the task.
5 Create page tables and external page tables in LSQA for the region.
Paging Supervisor
IEAPTCD
5.34
6.14, Step 6
Output
TCB
TCBSTI
TCBSCT
TCBRV

Diagram 6.17 Allocating a Nonspecific V=V Region (Module IEAVPRT0)

NOTES	ROUTINE NAME	LABEL
1 If the request is too large, GETAUX puts a return code of X'08' in register 15.	GETAUX	
2 An indication is passed to notify FBQSRCH to search from the low end of the free area.	FBQSRCH	
3	GETAUX	
4 GETPART initializes fields in the TCB: TCBSTI - Segment table index of the first segment TCBSCT - Number of segments assigned to the region TCBRV - Set to zero to indicate a pageable task	GETPART	
5 The Create Page Table subroutine of the paging supervisor creates the page tables and external page tables and updates the system and user segment table entries to point to the new page tables. The Create Page Table subroutine calls GETMAIN to build the tables. This call results in further processing as described in Diagram 6.4 "Allocating Storage for Control Blocks."	IEAPTCD	

Diagram 6.18
Allocating a Nonspecific V=R Region



1 Search FBQE queue for block of free virtual storage large enough for the region size requested.

2 If block of the exact size is not found, but a larger one is available

3 If there is no available block ≥ the request

4 Allocate page frames for the virtual region found.

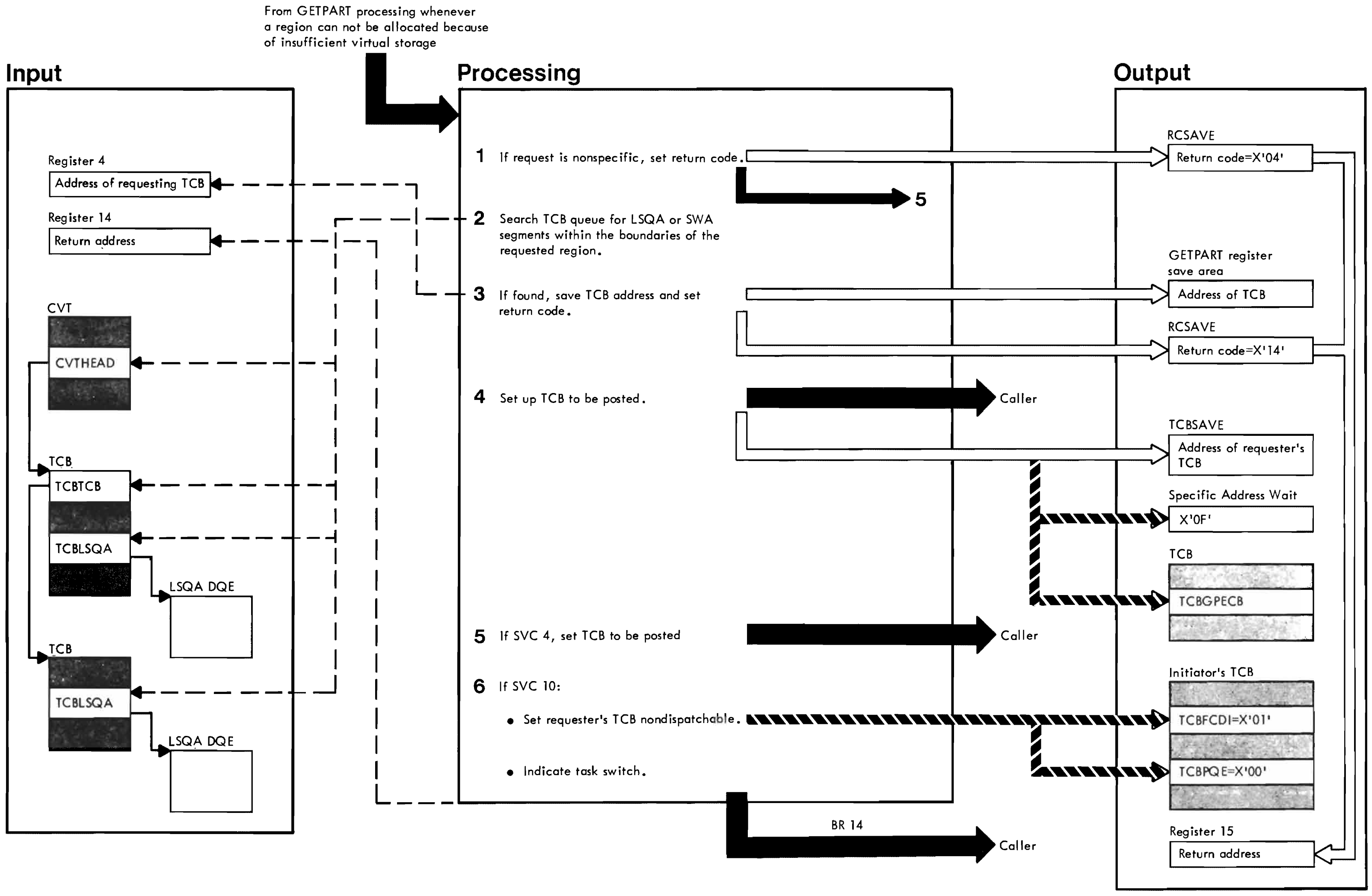
5 If an SQA, LSQA, or fixed page is located within the region, resume search.

6 Otherwise, save return code.

Diagram 6.18 Allocating a Nonspecific V=R Region (Module IEAVPRT0)

NOTES	ROUTINE NAME	LABEL
5 The V=R Region Allocation subroutine of the paging supervisor indicates this condition by a return code of X'10'.	IEAPVRAL IEAVPRT0	
6 Other return codes are; X'00' if the region has been successfully allocated, and X'0C' if the required page frames are not currently available, but are expected to become available soon.		

Diagram 6.19
Unsuccessful Virtual Region Allocation



1 If request is nonspecific, set return code.

2 Search TCB queue for LSQA or SWA segments within the boundaries of the requested region.

3 If found, save TCB address and set return code.

4 Set up TCB to be posted.

5 If SVC 4, set TCB to be posted

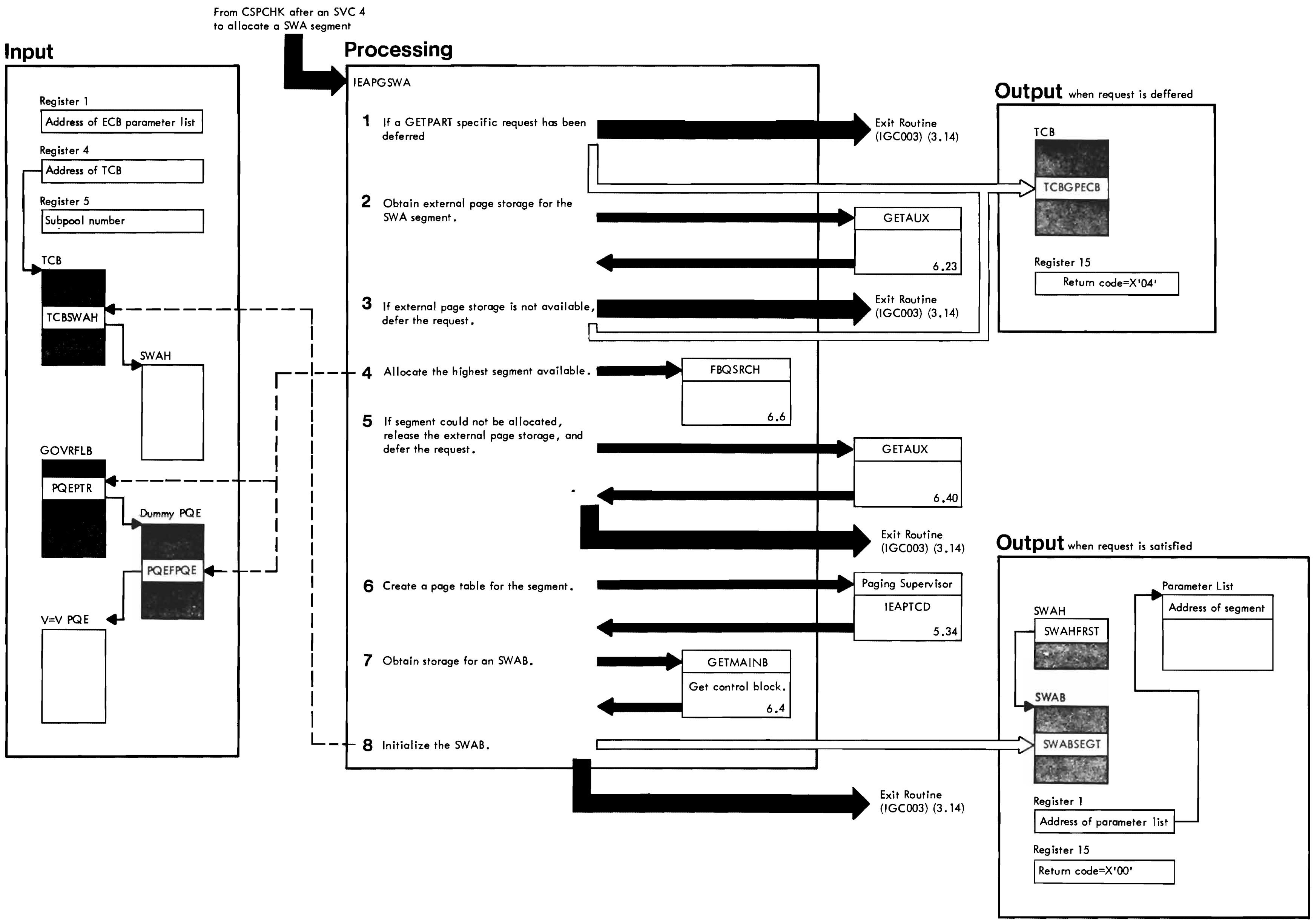
6 If SVC 10:

- Set requester's TCB nondispatchable.
- Indicate task switch.

Diagram 6.19 Unsuccessful Virtual Region Allocation (Module IEAVPRT0)

NOTES	ROUTINE NAME	LABEL
2 For GETPART specific requests, the routine examines the TCB chain (whose anchor is the CVTHEAD field of the CVT) to determine whether any LSQA or SWA segments are allocated within the boundaries of the requested region.	IEAVPRT0	

Diagram 6.20
Allocating a SWA Segment



1 If a GETPART specific request has been deferred

2 Obtain external page storage for the SWA segment.

3 If external page storage is not available, defer the request.

4 Allocate the highest segment available.

5 If segment could not be allocated, release the external page storage, and defer the request.

6 Create a page table for the segment.

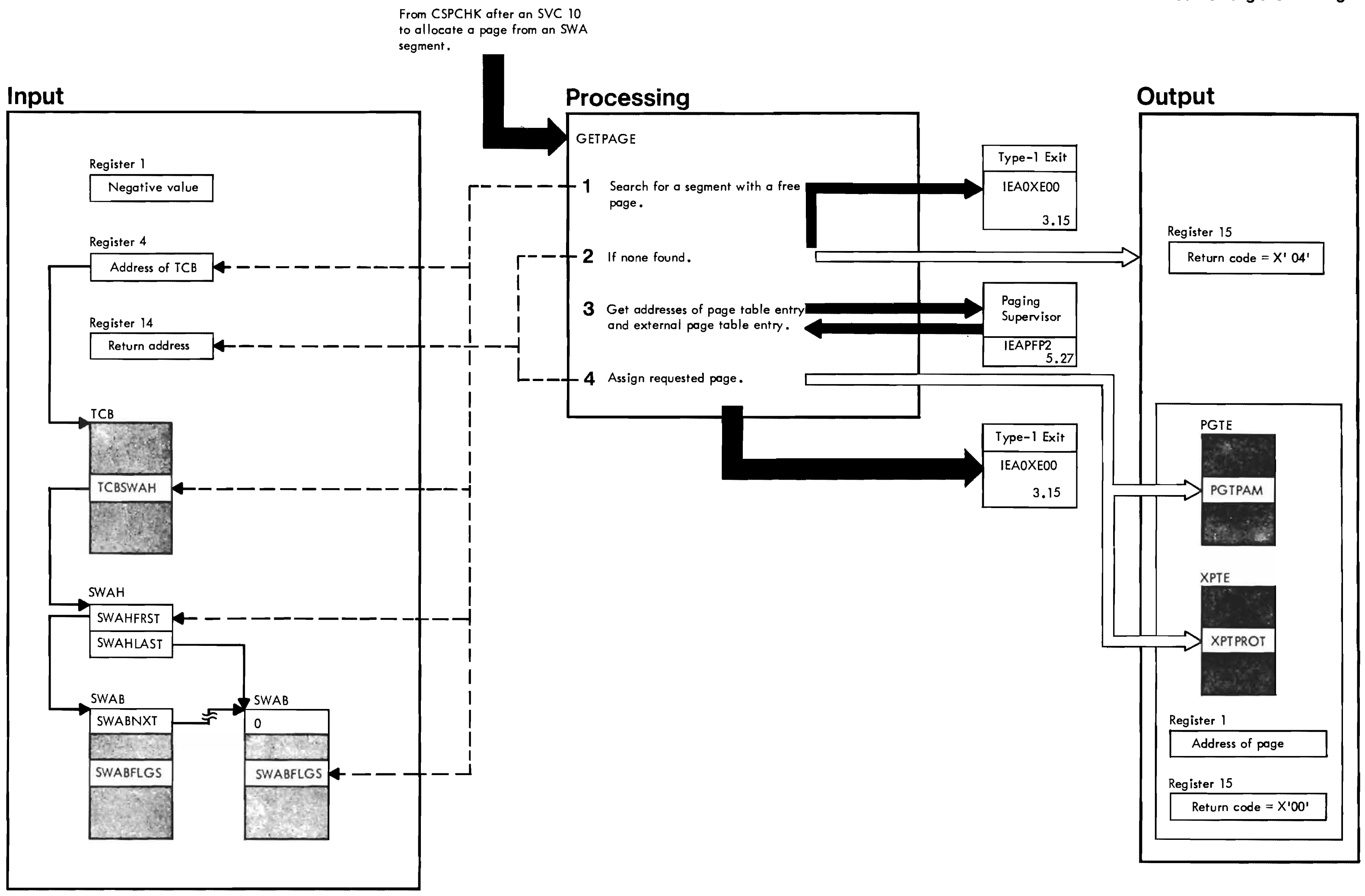
7 Obtain storage for an SWAB.

8 Initialize the SWAB.

Diagram 6.20 Allocating a SWA Segment (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
When the system issues an unconditional SVC 4 list instruction in ECB mode for a nonspecific single region to get a system work area, the GETMAIN routine CSPCHK gives control to IEAPGSWA to process the request.		
1 The routine determines whether a GETPART specific request is waiting to be honored by testing the WAITSW field in IEAVPRT0 for a value of X'0F'. If a request is waiting, the GETSWA request is deferred so a segment of virtual storage will not be assigned that could make it impossible to ever satisfy the specific request.	GETSWA	SETWAIT

Diagram 6.21
Allocating a SWA Page



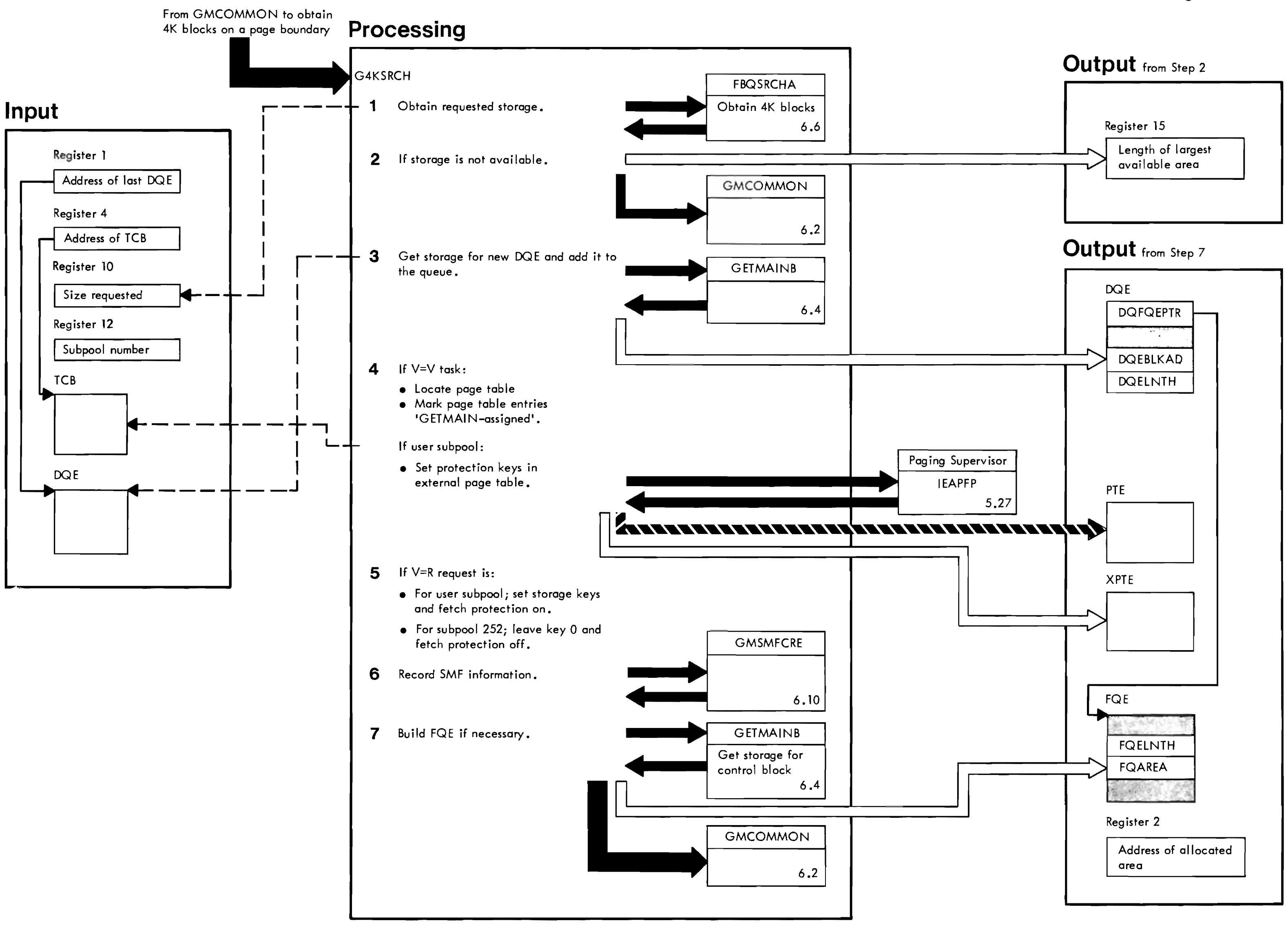
1 Search for a segment with a free page.

2 If none found.

3 Get addresses of page table entry and external page table entry.

4 Assign requested page.

Diagram 6.22
Allocating 4K or More



1 Obtain requested storage.

2 If storage is not available.

3 Get storage for new DQE and add it to the queue.

4 If V=V task:

- Locate page table
- Mark page table entries 'GETMAIN-assigned'.

If user subpool:

- Set protection keys in external page table.

5 If V=R request is:

- For user subpool; set storage keys and fetch protection on.
- For subpool 252; leave key 0 and fetch protection off.

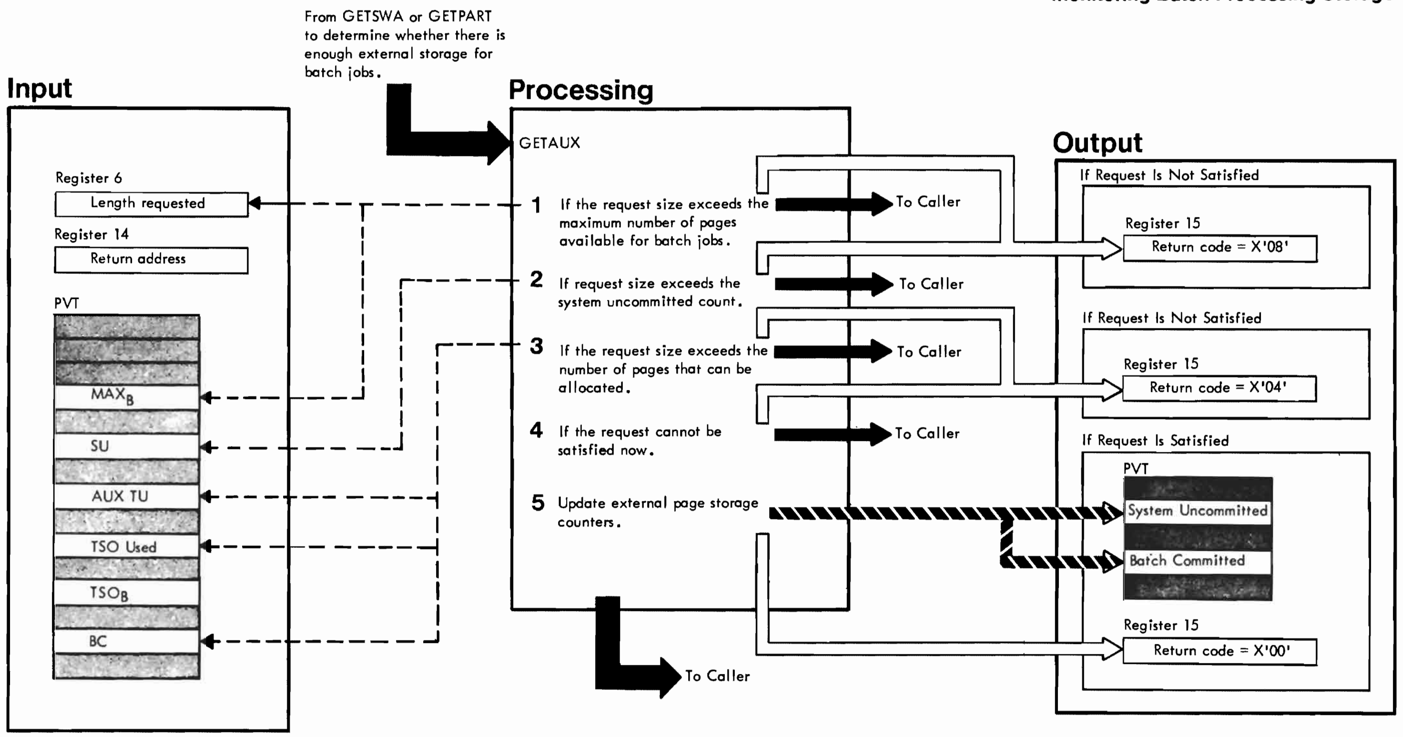
6 Record SMF information.

7 Build FQE if necessary.

Diagram 6.22 Allocating 4K or More (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
This routine is entered when a region request cannot be satisfied by GFRECORE, when a region request is greater than the maximum FQE size, or to satisfy a request for storage on a page boundary. This routine also sets the protection keys for the 4K blocks it obtains.		G4KSRCH
2 The value passed to GMCOMMON is the largest available size returned to G4KSRCH by FBQSRCHA unless GFRECORE was entered previously. GFRECORE determines the largest size available in the FQE queue. If GFRECORE was entered, G4KSRCH compares the values returned by GFRECORE and FBQSRCHA and passes the larger value to GMCOMMON.		
4 If the page table entry is in real storage, G4KSRCH sets the storage keys to the key of the TCB and sets fetch protection on.	IEAPFP2 G4KSRCH	
7 If storage on a page boundary was requested, the allocated area is taken from the low end of the block.		

Diagram 6.23
Monitoring Batch Processing Storage



From GETSWA or GETPART to determine whether there is enough external storage for batch jobs.

1 If the request size exceeds the maximum number of pages available for batch jobs.

2 If request size exceeds the system uncommitted count.

3 If the request size exceeds the number of pages that can be allocated.

4 If the request cannot be satisfied now.

5 Update external page storage counters.

Diagram 6.23 Monitoring Batch Processing Storage (Module IEAVPRTO)

NOTES	ROUTINE NAME	LABEL
1	GETPART	GETAUX
3 The request length is compared to the original total uncommitted (TU) number of pages minus the current number of pages committed to batch jobs (BC), minus the number of pages reserved to allow a TSO task to be forced from the system (TSC), minus the current number of pages in use for TSO (TSO_U). $TU - BC - TSO_B - TSC_U$		
4 The sum of the request length and the batch committed value is compared to the maximum batch committed value. If the sum is greater, the request cannot be honored now.		

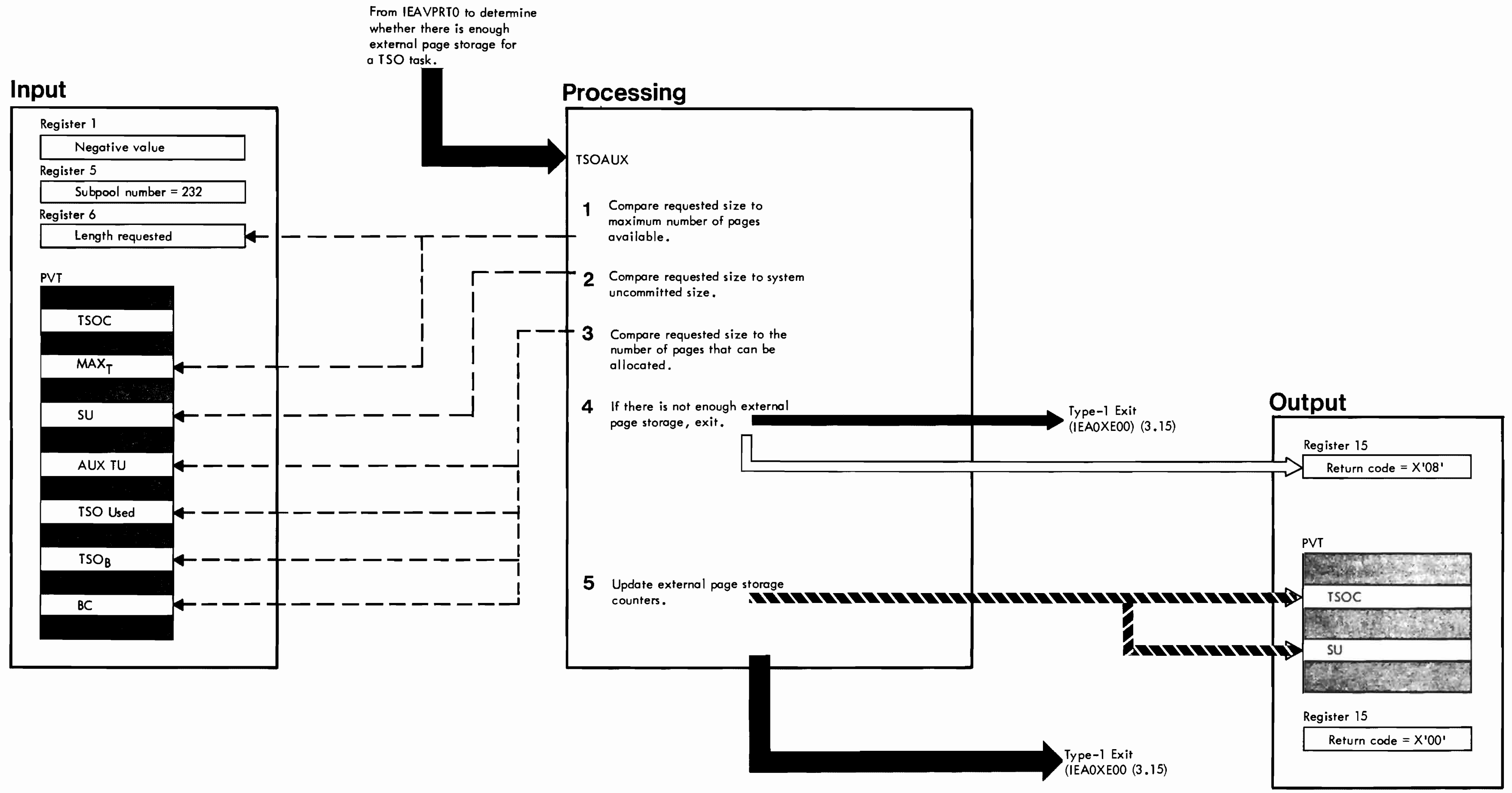

From IEAVPRT0 to determine whether there is enough external page storage for a TSO task.
Input
Register 1
Negative value
Register 5
Subpool number = 232
Register 6
Length requested
PVT
TSOC
MAX$_T$
SU
AUX TU
TSO Used
TSO$_B$
BC
Processing
TSOAUX
1 Compare requested size to maximum number of pages available.
2 Compare requested size to system uncommitted size.
3 Compare requested size to the number of pages that can be allocated.
4 If there is not enough external page storage, exit.
5 Update external page storage counters.
Type-1 Exit (IEAOXE00) (3.15)
Type-1 Exit (IEAOXE00 (3.15)
Output
Register 15
Return code = X'08'
PVT
TSOC
SU
Register 15
Return code = X'00'

Diagram 6.24 Monitoring TSO Storage (Module IEAVPRTO)

NOTES	ROUTINE NAME	LABEL
3 The requested size is compared to the original total uncommitted (TU) number of pages minus the current number of pages committed to batch processing (BC) minus the number of pages reserved to allow a TSO task to be forced from the system (TSO_B) minus the current number of pages in use for TSO (TSO_U). $TU - BC - TSO_B - TSO_U$	GETPART	TSOAUX
4 The routine exits to the Type 1 Exit routine with a return code of X'08' if: request size > maximum number of pages (MAX_T) request size > system uncommitted pages (SU) request size > $TU - BC - TSO_B - TSO_U$		
5 The system uncommitted count is decremented and the TSO committed count is incremented by the number of bytes requested.		

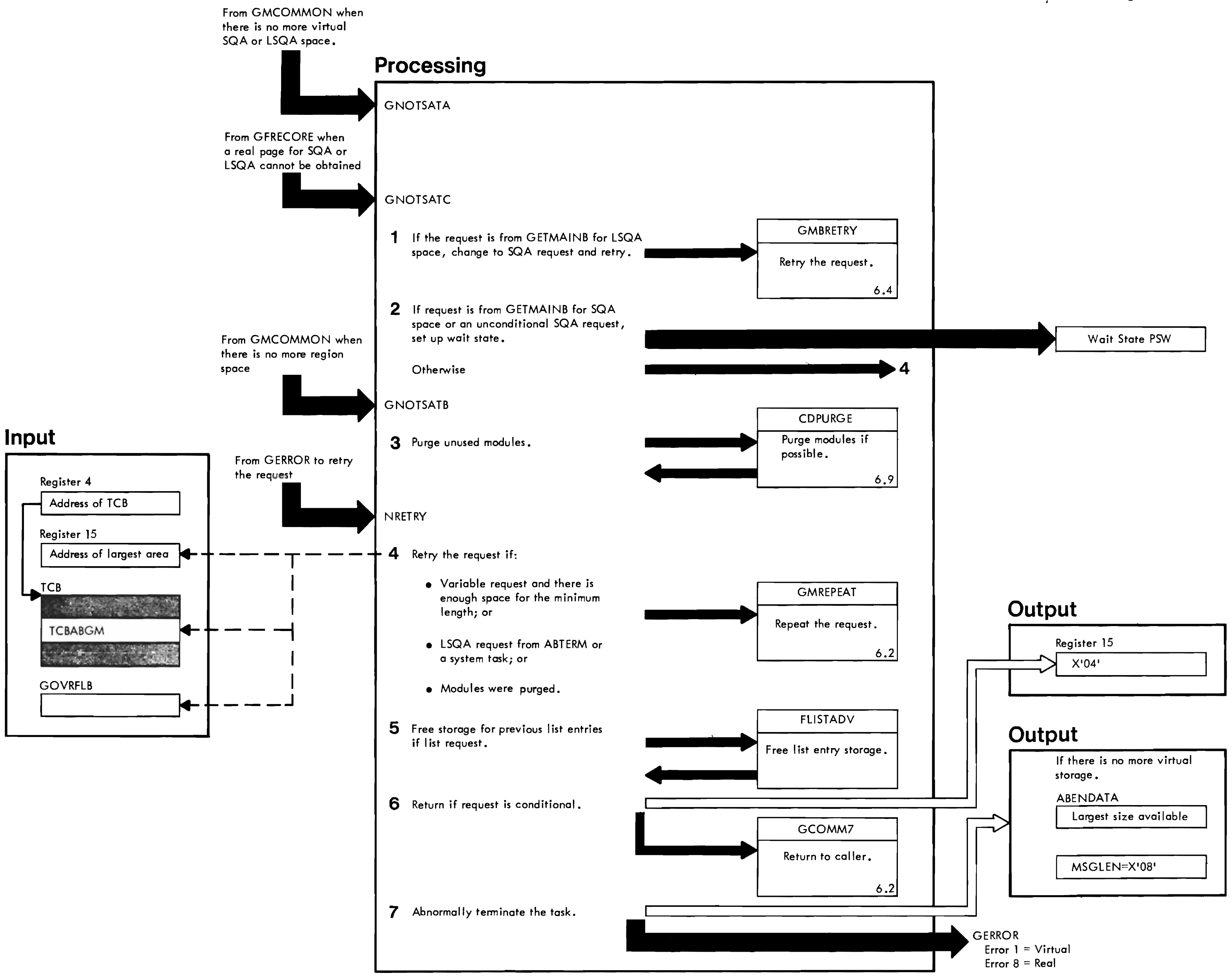
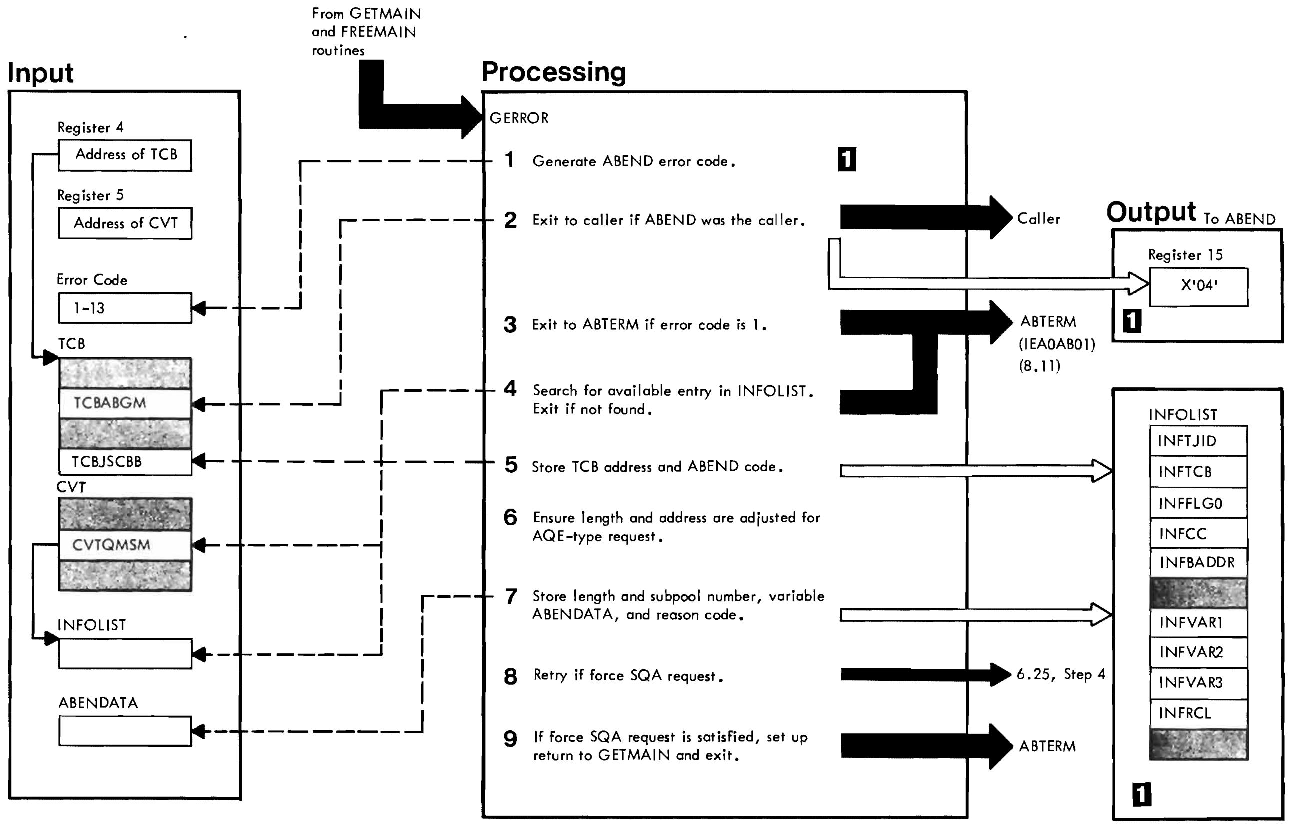

From GMCOMMON when there is no more virtual SQA or LSQA space.
Processing
GNOTSATA
From GFRECORE when a real page for SQA or LSQA cannot be obtained
GNOTSATC
1 If the request is from GETMAINB for LSQA space, change to SQA request and retry.
GMBRETRY
Retry the request.
6.4
2 If request is from GETMAINB for SQA space or an unconditional SQA request, set up wait state.
Wait State PSW
Otherwise
4
From GMCOMMON when there is no more region space
GNOTSATB
3 Purge unused modules.
CDPURGE
Purge modules if possible.
6.9
Input
Register 4
Address of TCB
Register 15
Address of largest area
TCB
TCBABGM
GOVRFLB
From GERROR to retry the request
NRETRY
4 Retry the request if:
• Variable request and there is enough space for the minimum length; or
• LSQA request from ABTERM or a system task; or
• Modules were purged.
GMREPEAT
Repeat the request.
6.2
Output
Register 15
X'04'
5 Free storage for previous list entries if list request.
FLISTADV
Free list entry storage.
Output
If there is no more virtual storage.
ABENDATA
Largest size available
MSGLEN=X'08'
6 Return if request is conditional.
GCOMM7
Return to caller.
6.2
7 Abnormally terminate the task.
GERROR
Error 1 = Virtual
Error 8 = Real

Diagram 6.25 Out-of-Storage Processing (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
4 If the request is an LSQA request during abnormal termination, or if the request is from a system task, the SPQE for the SQA (GOVRFLB) is used to retry the request. If the request is for SQA or LSQA, it may be necessary to repeat it more than once because the largest available space may cross a page boundary and cannot therefore be completely allocated.	GNOTSAT	

• Diagram 6.26
Error Processing for GETMAIN/FREEMAIN



1 Generate ABEND error code.

2 Exit to caller if ABEND was the caller.

3 Exit to ABTERM if error code is 1.

4 Search for available entry in INFOLIST. Exit if not found.

5 Store TCB address and ABEND code.

6 Ensure length and address are adjusted for AQE-type request.

7 Store length and subpool number, variable ABENDATA, and reason code.

8 Retry if force SQA request.

9 If force SQA request is satisfied, set up return to GETMAIN and exit.

• Diagram 6.26 Error Processing for GETMAIN/FREEMAIN (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
3 The ABNDATA area is cleared. There are no messages written for code 1 because it is self-explanatory.	GERROR	
5 If this was a branch entry, INFFLG0 is set and the caller's return address is placed in the information list (INFBADDR). If the TCBJSCBB field is nonzero, the JSCBTJID field is moved into the information list at INFTJID.		
6 For AQE-type requests, eight is subtracted from the length and a check is made to see if the address has been adjusted. (There are times when the length and address adjustments are not synchronized.) If it has been, eight is added to the address to compensate for the AQE.		
7 For FREEMAIN requests, the FREEMAIN address is also placed in INFVAR2.		
8		NRETRY
1 Error exits to the ABTERM routine cause the task that issued the request to be abnormally terminated. Register 1 contains the error code. Register 3 contains the address of the CVT. Register 4 contains the address of the TCB. Register 14 contains the address of IEA0XE00.		

Input to GERROR	Output from GERROR
ERROR CODE= 1	X'0104'
	X'010A'
ERROR CODE= 2	X'020A'
ERROR CODE= 3	X'0305'
	X'030A'
ERROR CODE= 4	X'040A'
ERROR CODE= 5	X'0504'
	X'0505'
ERROR CODE= 6	X'0604'
	X'0605'
ERROR CODE= 8	X'0804'
	X'080A'
ERROR CODE= 9	X'0905'
	X'090A'
ERROR CODE= 10	X'0A05'
	X'0A0A'
ERROR CODE= 11	X'0B04'
	X'0B05'
	X'0B0A'
ERROR CODE= 13	X'0D05'
	X'0D0A'

Diagram 6.27
Overview of Releasing Storage

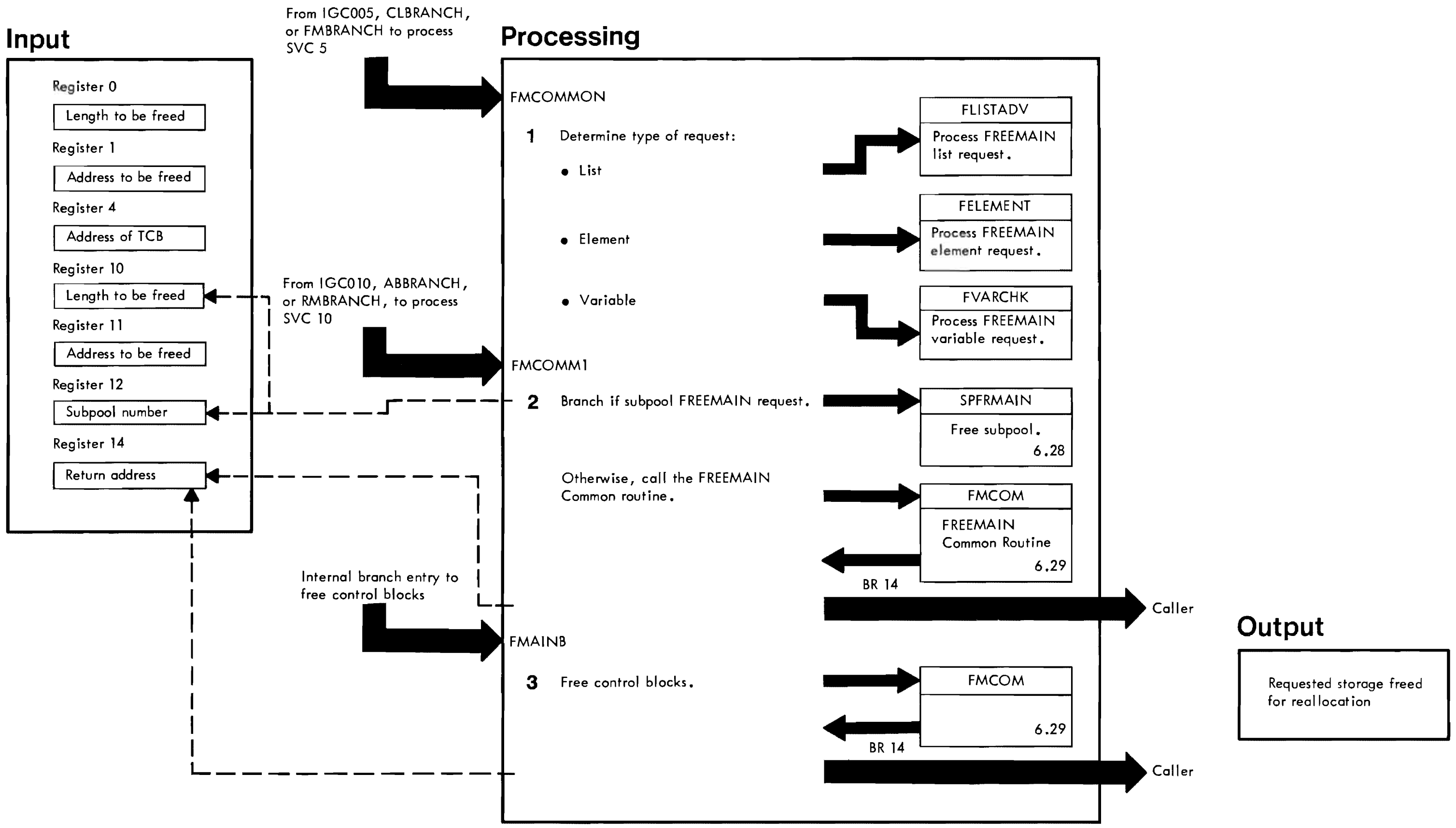

Input
Register 0
Length to be freed
Register 1
Address to be freed
Register 4
Address of TCB
Register 10
Length to be freed
Register 11
Address to be freed
Register 12
Subpool number
Register 14
Return address
From IGC005, CLBRANCH, or FMBRANCH to process SVC 5
From IGC010, ABBRANCH, or RMBRANCH, to process SVC 10
Internal branch entry to free control blocks
Processing
FMCOMMON
1 Determine type of request:
• List
• Element
• Variable
FLISTADV
Process FREEMAIN list request.
FELEMENT
Process FREEMAIN element request.
FVARCHK
Process FREEMAIN variable request.
FMCOMM1
2 Branch if subpool FREEMAIN request.
SPFRMAIN
Free subpool. 6.28
Otherwise, call the FREEMAIN Common routine.
FMCOM
FREEMAIN Common Routine 6.29
BR 14
Caller
FMAINB
3 Free control blocks.
FMCOM
6.29
BR 14
Caller
Output
Requested storage freed for reallocation

Diagram 6.27 Overview of Releasing Storage (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 FLISTADV - This routine processes FREEMAIN list requests. For the first entry in the list, it calls FMCOM. For all subsequent entries in the list, it calls FCOM1 to bypass the subpool check processing. When the last entry in the list has been processed, it branches to FRETRN1 in FMCOMM1. FELEMENT - This routine picks up the element length entry, sets up registers as though the request were an SVC 10 request, and calls FMCOMM1A. FVARCHK - This routine simply picks up the length from the second address entry, sets up registers as though the request were an SVC 10 request, and calls FMCOMM1A.	FMCOMMON FLISTADV FELEMENT FVARCHK	

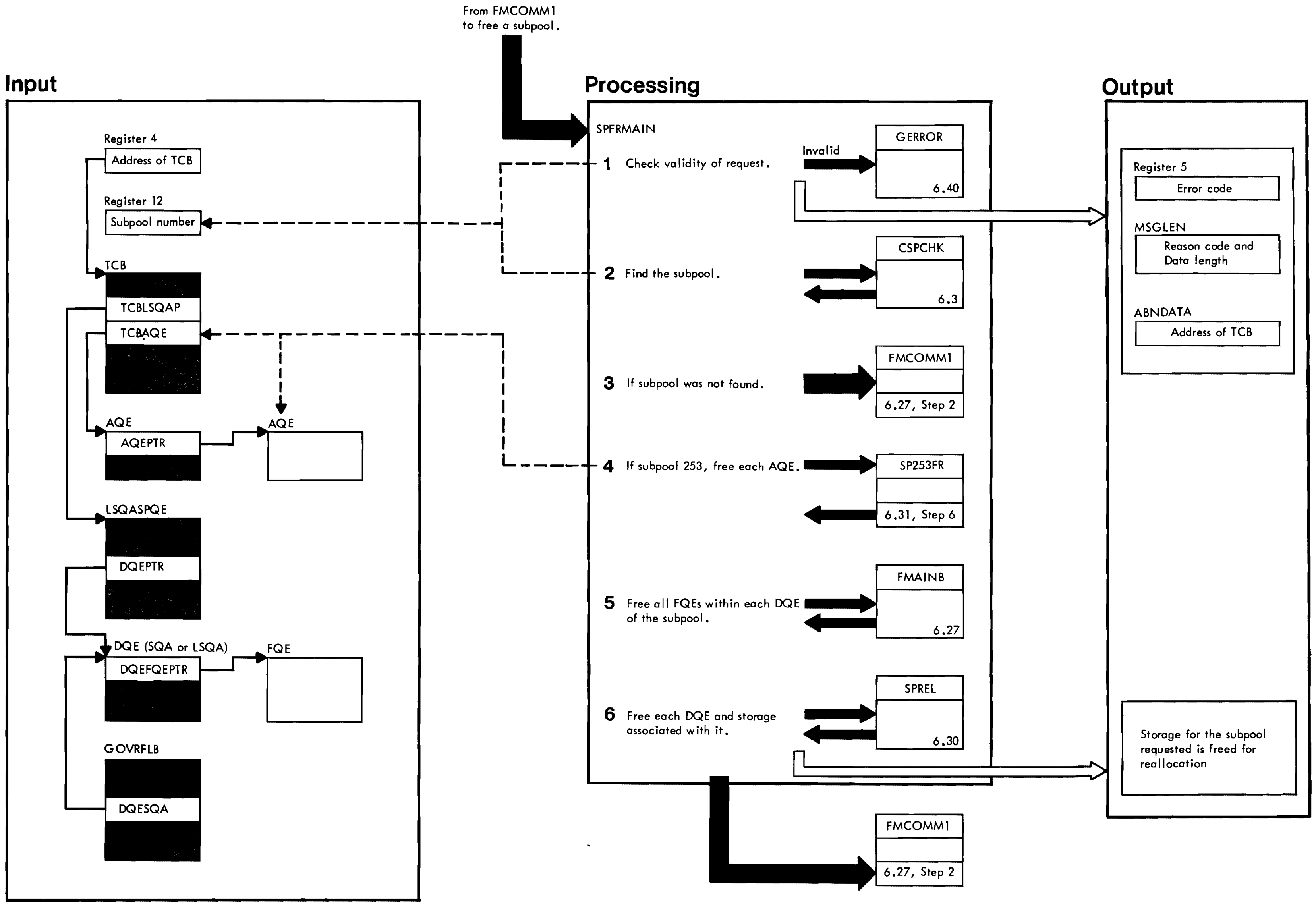
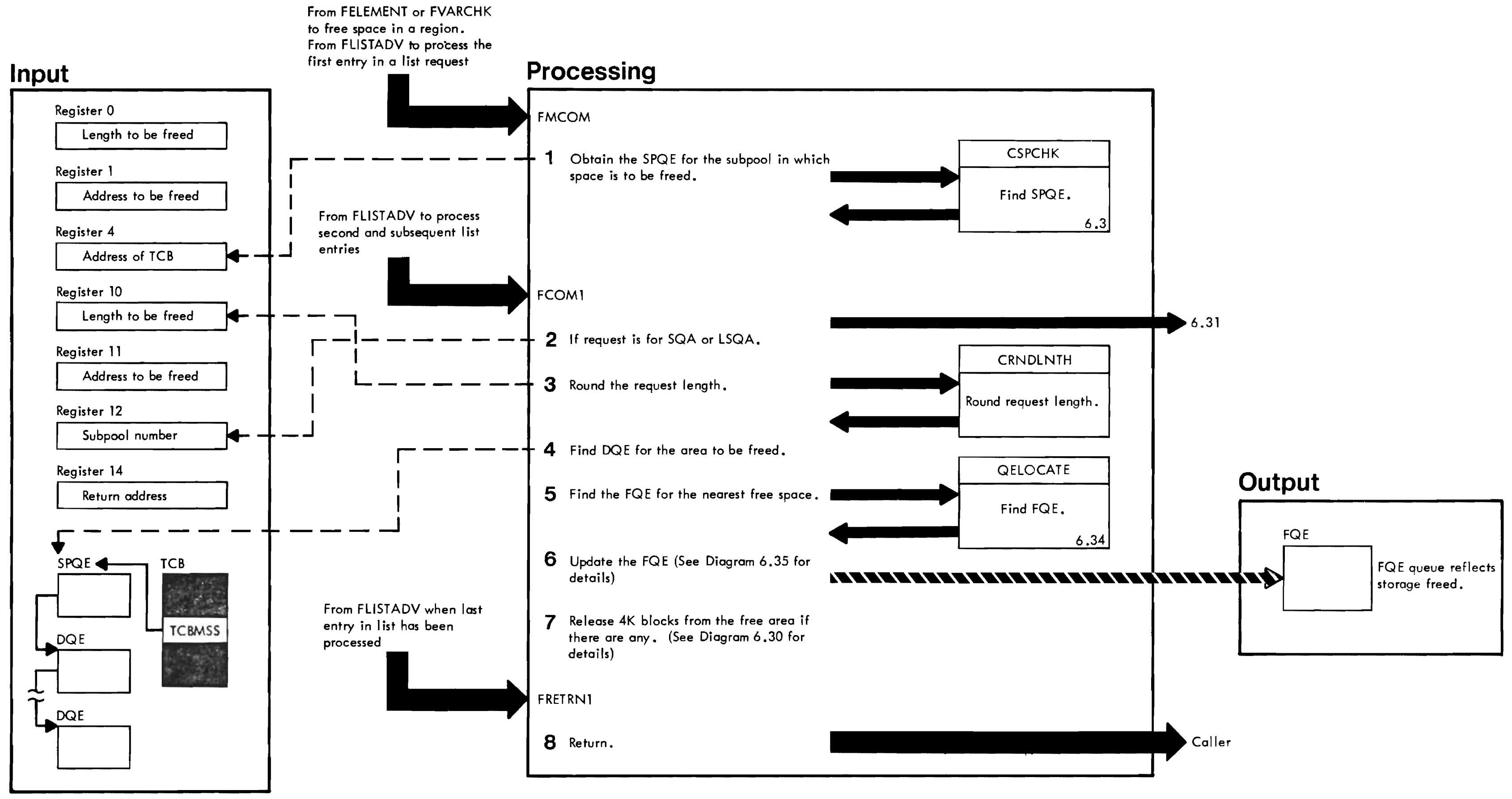

From FMCOMM1 to free a subpool.
Input
Register 4
Address of TCB
Register 12
Subpool number
TCB
TCBLSQAP
TCBAQE
AQE
AQEPTR
AQE
LSQASPQE
DQEPTR
DQE (SQA or LSQA)
DQEFQEPTR
FQE
GOVRFLB
DQESQA
Processing
SPFRMAIN
1 Check validity of request.
Invalid
GERROR
6.40
2 Find the subpool.
CSPCHK
6.3
3 If subpool was not found.
FMCOMM1
6.27, Step 2
4 If subpool 253, free each AQE.
SP253FR
6.31, Step 6
5 Free all FQEs within each DQE of the subpool.
FMAINB
6.27
6 Free each DQE and storage associated with it.
SPREL
6.30
FMCOMM1
6.27, Step 2
Output
Register 5
Error code
MSGLEN
Reason code and Data length
ABNDATA
Address of TCB
Storage for the subpool requested is freed for reallocation

Diagram 6.28 Freeing a Subpool (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 If the length is not zero, a reason code of X'08' and data length of X'0C' are set in MSGLEN and GERROR is entered with an error code of 3. If this is a request to free subpool 243, 244, 245, or 255, a reason code of X'08' are set in MSGLEN, and GERROR is entered with an error code of 4.	SPFRMAIN GERROR	
2 If the subpool is 0 and the requester is not in key 0, processing is the same as for error code 4 in Step 1.		



1 Obtain the SPQE for the subpool in which space is to be freed.

2 If request is for SQA or LSQA.

3 Round the request length.

4 Find DQE for the area to be freed.

5 Find the FQE for the nearest free space.

6 Update the FQE (See Diagram 6.35 for details)

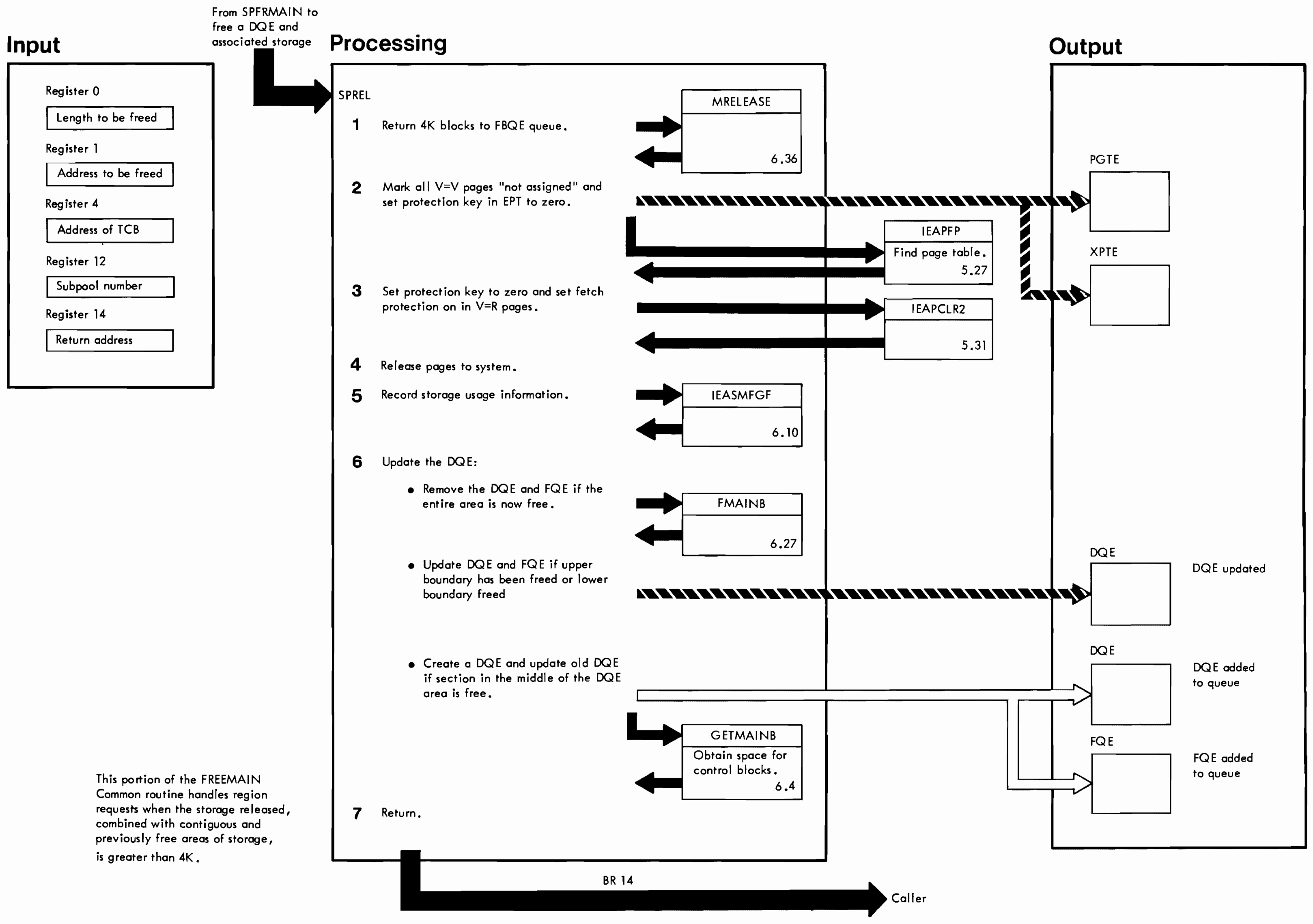
7 Release 4K blocks from the free area if there are any. (See Diagram 6.30 for details)

8 Return.

Diagram 6.29 Freeing Storage within a Region (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
3 The CRNDLNTH subroutine rounds the request length to an eight-byte multiple.	CRNDLNTH	
7	FCOM	SPREL

Diagram 6.30
Freeing More Than 4K



Input

Register 0
Length to be freed
Register 1
Address to be freed
Register 4
Address of TCB
Register 12
Subpool number
Register 14
Return address

From SPFRMAIN to free a DQE and associated storage

Processing

SPREL

1 Return 4K blocks to FBQE queue.

2 Mark all V=V pages "not assigned" and set protection key in EPT to zero.

3 Set protection key to zero and set fetch protection on in V=R pages.

4 Release pages to system.

5 Record storage usage information.

6 Update the DQE:

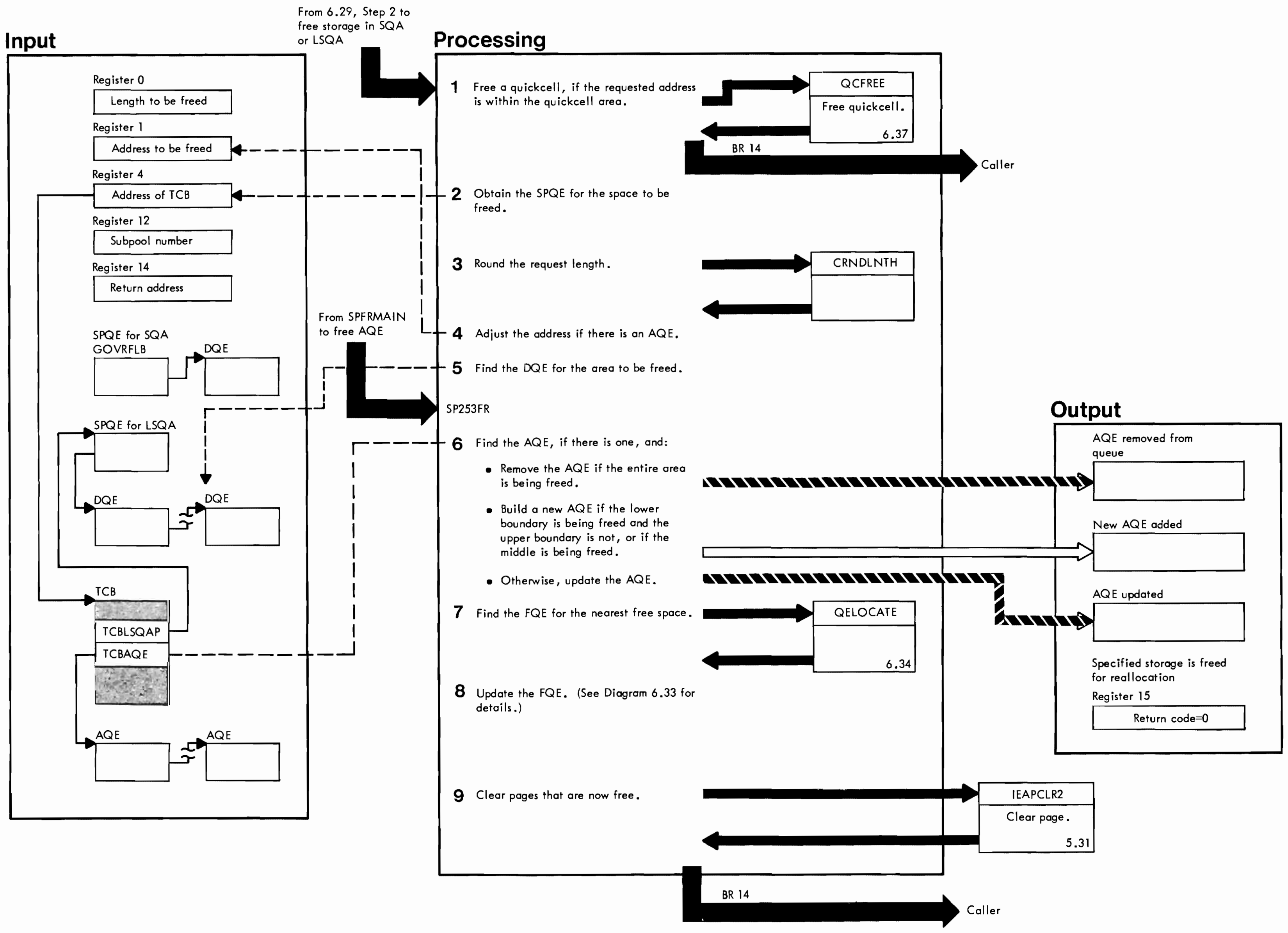
- Remove the DQE and FQE if the entire area is now free.
- Update DQE and FQE if upper boundary has been freed or lower boundary freed
- Create a DQE and update old DQE if section in the middle of the DQE area is free.

7 Return.

Output

This portion of the FREEMAIN Common routine handles region requests when the storage released, combined with contiguous and previously free areas of storage, is greater than 4K.

Diagram 6.31
Freeing Storage in the SQA or an LSQA

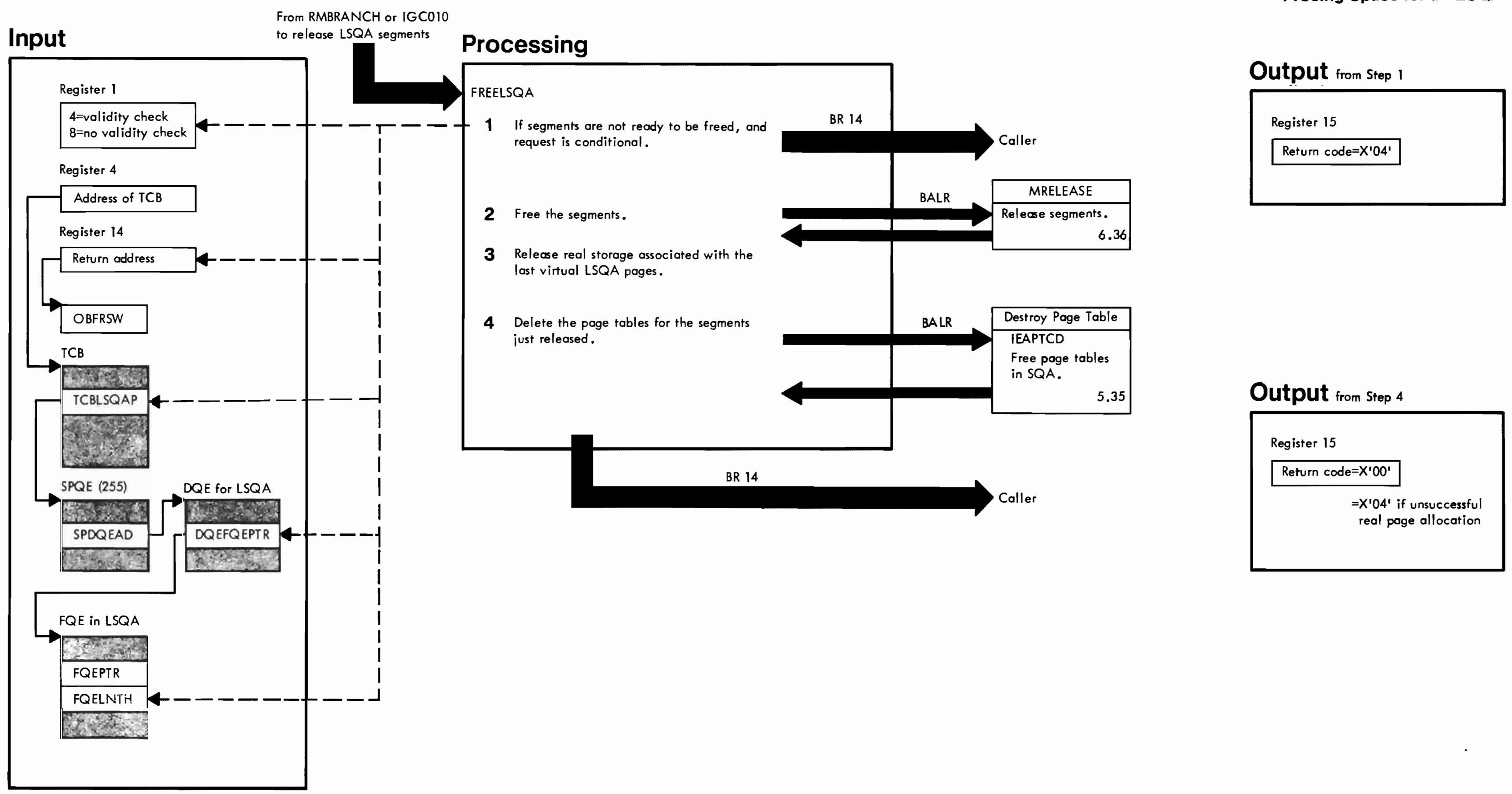


Input
Register 0
Length to be freed
Register 1
Address to be freed
Register 4
Address of TCB
Register 12
Subpool number
Register 14
Return address
SPQE for SQA GOVRFLB
DQE
SPQE for LSQA
DQE
DQE
TCB
TCBLSQAP
TCBAQE
AQE
AQE
From 6.29, Step 2 to free storage in SQA or LSQA
From SPFRMAIN to free AQE
Processing
1 Free a quickcell, if the requested address is within the quickcell area.
QCFREE
Free quickcell.
6.37
BR 14
Caller
2 Obtain the SPQE for the space to be freed.
3 Round the request length.
CRNDLNTH
4 Adjust the address if there is an AQE.
5 Find the DQE for the area to be freed.
SP253FR
6 Find the AQE, if there is one, and:
• Remove the AQE if the entire area is being freed.
• Build a new AQE if the lower boundary is being freed and the upper boundary is not, or if the middle is being freed.
• Otherwise, update the AQE.
7 Find the FQE for the nearest free space.
QELOCATE
6.34
8 Update the FQE. (See Diagram 6.33 for details.)
9 Clear pages that are now free.
IEAPCLR2
Clear page.
5.31
BR 14
Caller
Output
AQE removed from queue
New AQE added
AQE updated
Specified storage is freed for reallocation
Register 15
Return code=0

Diagram 6.31 Freeing Storage in the SQA or an LSQA (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 If there is a quickcell descriptor block and the address of the space to be freed is in the quickcell area, the QCFREE subroutine is used to free the quickcell.	FCOM QCFREE	
2 A GETMAIN request for LSQA may have been satisfied from the SQA. If the address to be freed is within the SQA, the SPQE for SQA (GOVRFLB) is used.	FCOM1	
3 If the address to be freed is not on a doubleword boundary, if the request was an SVC 5, a return code of 0 and variable length of X'0C' are placed in MSGLEN or if it was not an SVC 5 request, a reason code of 0 and variable length of X'08' are placed in MSGLEN. Then the task is abnormally terminated by going to GERROR with a code of 9. Otherwise, the CRNDLNTH routine rounds the request length to an eight-byte multiple and adds eight bytes to the length if there is an AQE.		CRNDLNTH
4 Subpools 243, 244, 253, and 254 are for storage described by allocated queue elements (AQEs). When storage is freed from one of these subpools, the start address must be backed up eight bytes to free the AQE.		FCOM
6 The AQE is pointed to by the TCBAQE field of the TCB. If there is no AQE chain for subpool 243, 244, 253, or 254 requests, the requesting task is abnormally terminated (GERROR code 13). When the AQE is obtained it is updated: If the whole area described by the AQE is being freed, the AQE is removed from the queue. If the upper boundary of the AQE is not being freed, eight is subtracted from the length of the area to be freed to allow space for a new AQE. The new AQE is built and added to the queue. If the lower boundary of the area described by the original AQE is not being freed, the length of the AQE is changed. If the lower boundary of the AQE is being freed, the AQE is removed from the queue. If the lower boundary is not being freed and the upper boundary is being freed, eight is substracted from the length and added to the address of the area to be freed.		
9 The Clear Page call is skipped if NIP is in process.		

Diagram 6.32
Freeing Space for an LSQA



1 If segments are not ready to be freed, and request is conditional.

2 Free the segments.

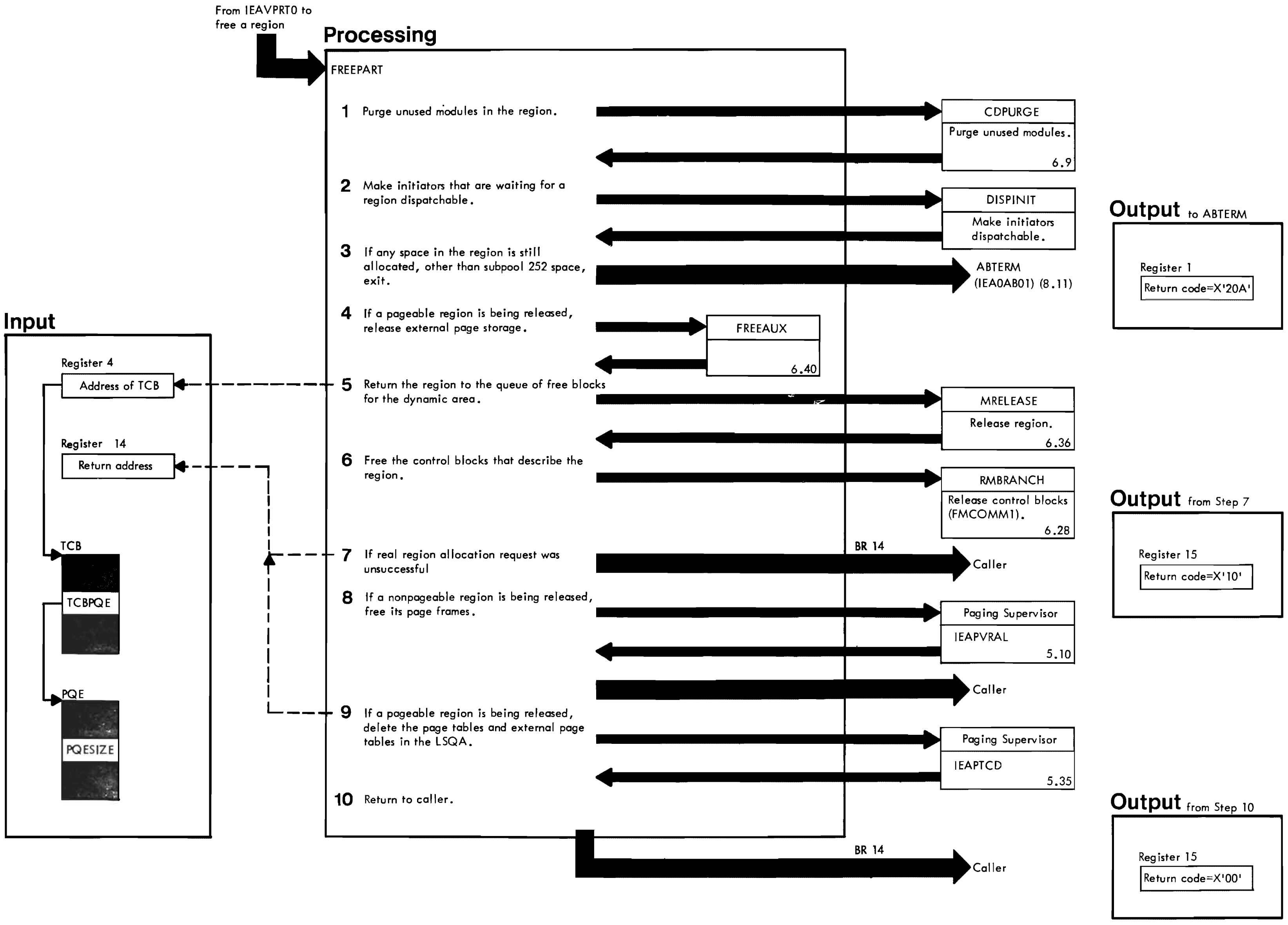
3 Release real storage associated with the last virtual LSQA pages.

4 Delete the page tables for the segments just released.

Diagram 6.32 Freeing Space for an LSQA (Module IEAPLSQA)

NOTES	ROUTINE NAME	LABEL
1 If there is more than one FQE, or if the FQE for the LSQA describes a space smaller than the size of the segments when they were first allocated, the segments cannot be freed. If register 1 equals eight, these checks are skipped.	IEAPLSQA	FREELSQA

Diagram 6.33
Freeing Space for a Region



From IEAVPRT0 to free a region

Processing

FREEPART

1 Purge unused modules in the region.

2 Make initiators that are waiting for a region dispatchable.

3 If any space in the region is still allocated, other than subpool 252 space, exit.

4 If a pageable region is being released, release external page storage.

5 Return the region to the queue of free blocks for the dynamic area.

6 Free the control blocks that describe the region.

7 If real region allocation request was unsuccessful

8 If a nonpageable region is being released, free its page frames.

9 If a pageable region is being released, delete the page tables and external page tables in the LSQA.

10 Return to caller.

Input

Register 4: Address of TCB

Register 14: Return address

TCB: TCBPQE

PQE: PQESIZE

CDPURGE — Purge unused modules. 6.9

DISPINIT — Make initiators dispatchable.

ABTERM (IEA0AB01) (8.11)

FREEAUX 6.40

MRELEASE — Release region. 6.36

RMBRANCH — Release control blocks (FMCOMM1). 6.28

BR 14 Caller

Paging Supervisor IEAPVRAL 5.10

Caller

Paging Supervisor IEAPTCD 5.35

BR 14 Caller

Output to ABTERM

Register 1: Return code=X'20A'

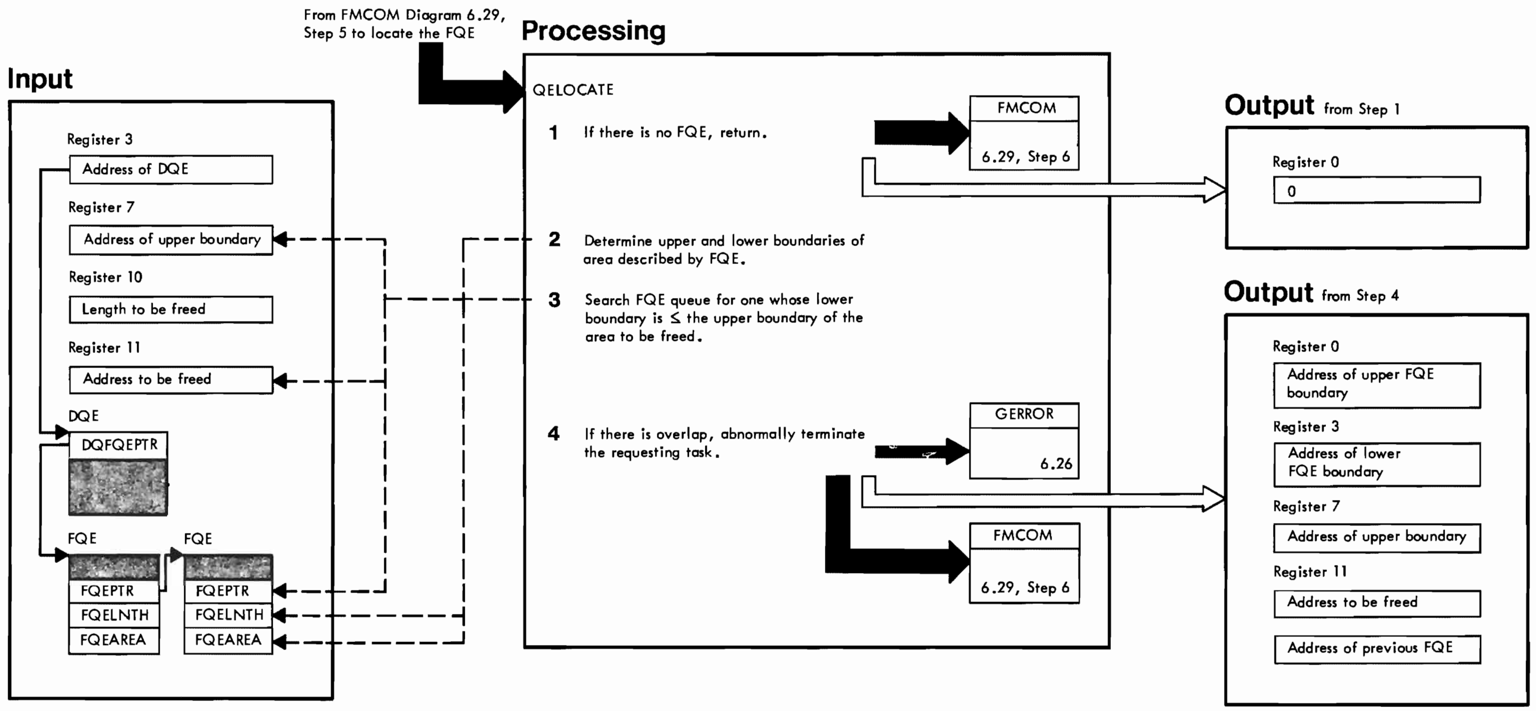
Output from Step 7

Register 15: Return code=X'10'

Output from Step 10

Register 15: Return code=X'00'

Diagram 6.34
Locating an FQE to Free Space



1 If there is no FQE, return.

2 Determine upper and lower boundaries of area described by FQE.

3 Search FQE queue for one whose lower boundary is ≤ the upper boundary of the area to be freed.

4 If there is overlap, abnormally terminate the requesting task.

Diagram 6.34 Locating an FQE to Free Space (Module IEAVGM00)

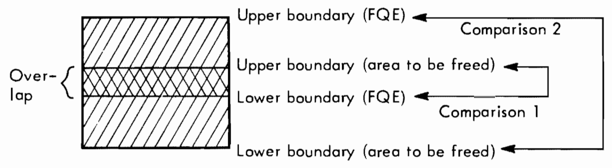
NOTES	ROUTINE NAME	LABEL
2 For a region FQE, the upper boundary is the FQEAREA. For an LSQA or SQA FQE, eight is added to the address to include the FQE.	QELOCATE	
3 The lower boundary is calculated by subtracting the FQE length from the upper boundary.		
4 The area to be freed overlaps a free area if the FQE lower boundary is less than the upper boundary to be freed and the FQE upper boundary is greater than the beginning address of the area to be freed.  Upper boundary (FQE) Comparison 2 Over-lap Upper boundary (area to be freed) Lower boundary (FQE) Comparison 1 Lower boundary (area to be freed)		

Diagram 6.35
Updating an FQE to Free Space

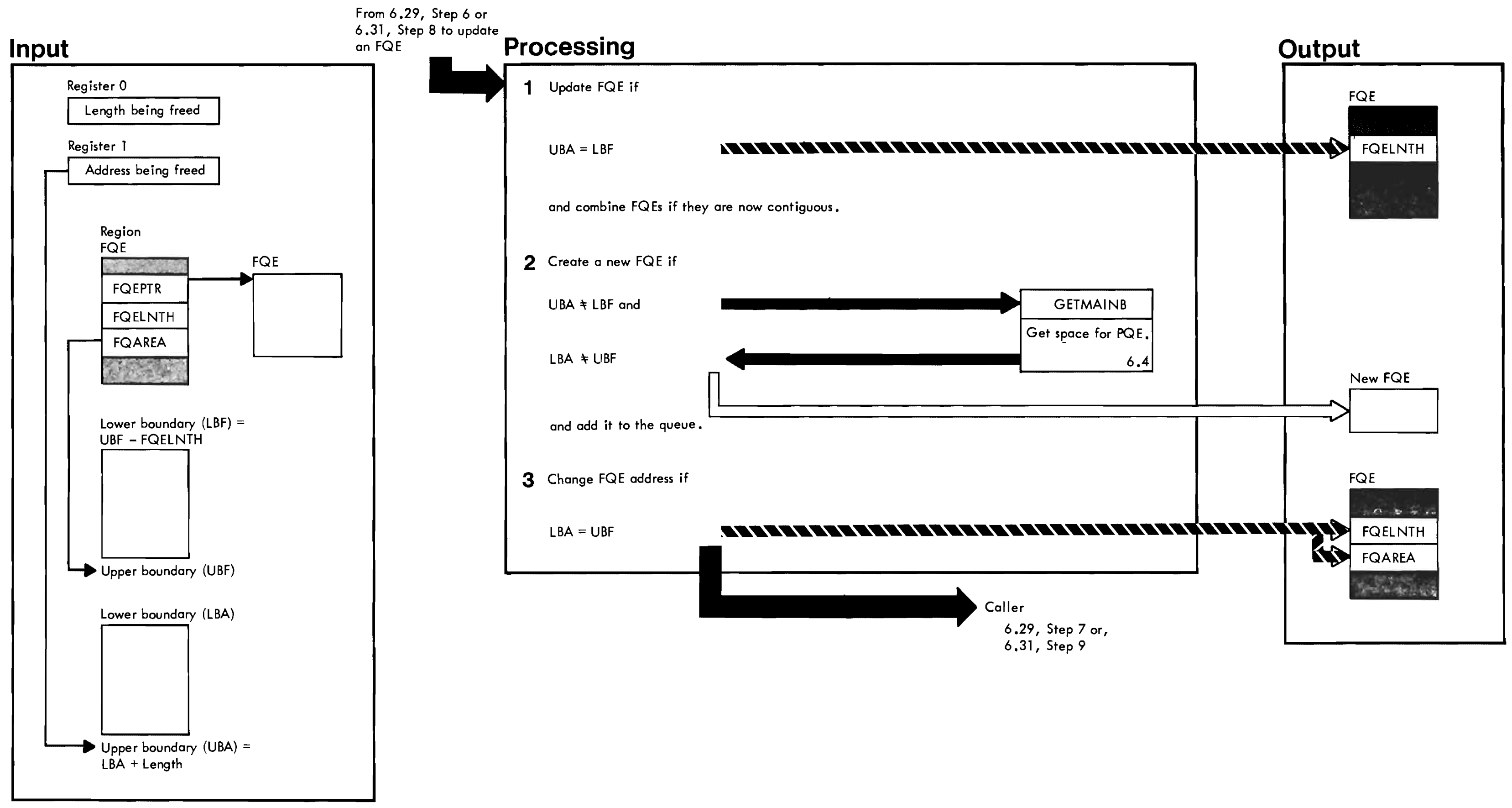
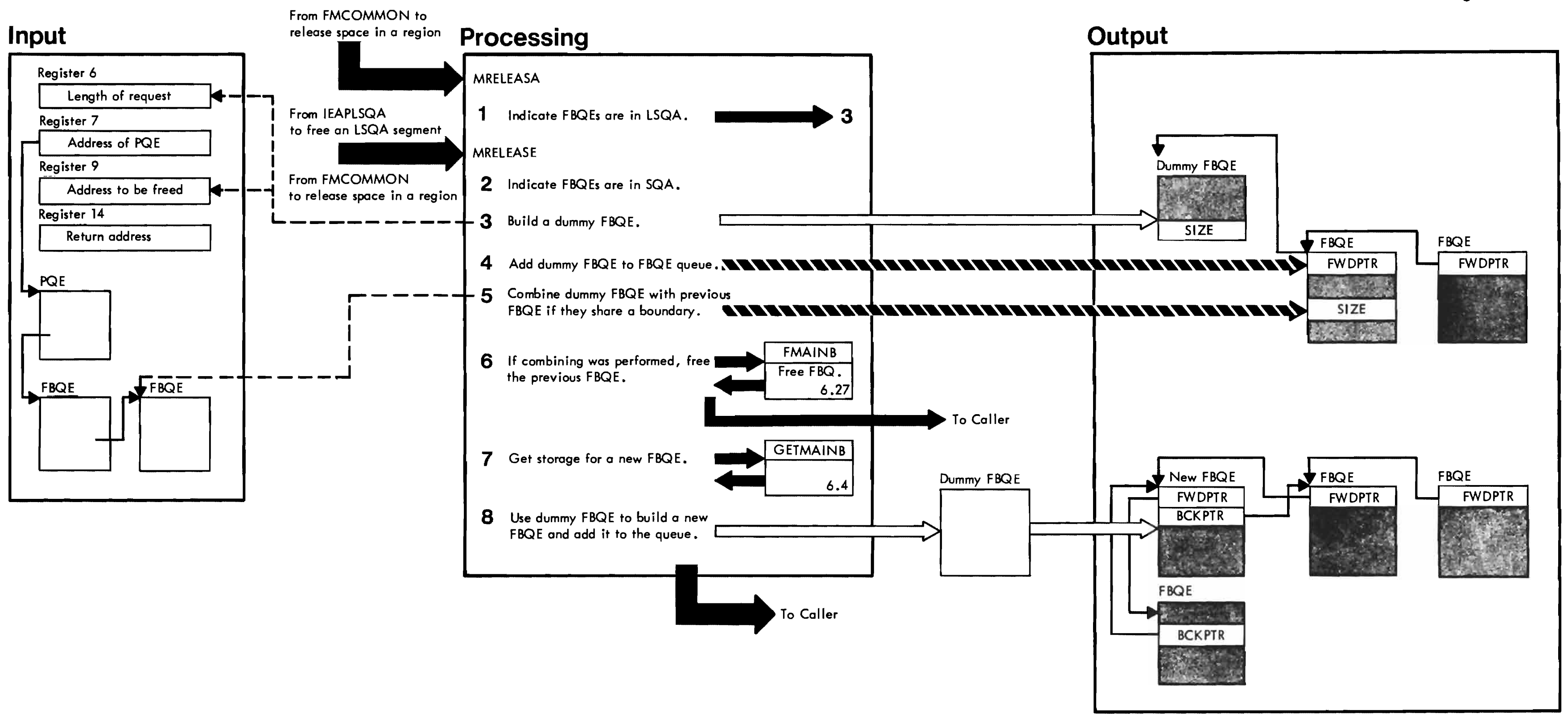


Diagram 6.35 Updating an FQE to Free Storage Space (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 If the upper boundary of the area being freed (UBA) equals the lower boundary of the FQE (LBF), the length of the FQE is increased by the length of the area being freed.	FCOM	
2 If the upper boundary of the area to be freed is not equal to the lower boundary of the area described by the FQE and the upper boundary of the FQE is not equal to the lower boundary of the area being freed, a new FQE must be created. (For a region request to free the entire area described by the DQE, a new FQE is not created since it would be freed in later processing of this request.)		
3 If the upper boundary of the FQE equals the lower boundary of the area being freed, the FQE is moved if it is an SQA or LSQA FQE or its address is changed if it is a region FQE. The length of the FQE and the length being freed are combined to become the new length.		

Diagram 6.36
Freeing 4K Blocks



MRELEASA

1 Indicate FBQEs are in LSQA. → 3

MRELEASE

2 Indicate FBQEs are in SQA.

3 Build a dummy FBQE.

4 Add dummy FBQE to FBQE queue.

5 Combine dummy FBQE with previous FBQE if they share a boundary.

6 If combining was performed, free the previous FBQE.

7 Get storage for a new FBQE.

8 Use dummy FBQE to build a new FBQE and add it to the queue.

Diagram 6.36 Freeing 4K Blocks (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
3 A dummy FBQE (RBLOCK) is built in a work area and initialized with the input parameters in registers 6 and 9.		MRELEASE
4 The address of the dummy FBQE is compared to the address in the FBQE to determine where it belongs in the queue.		

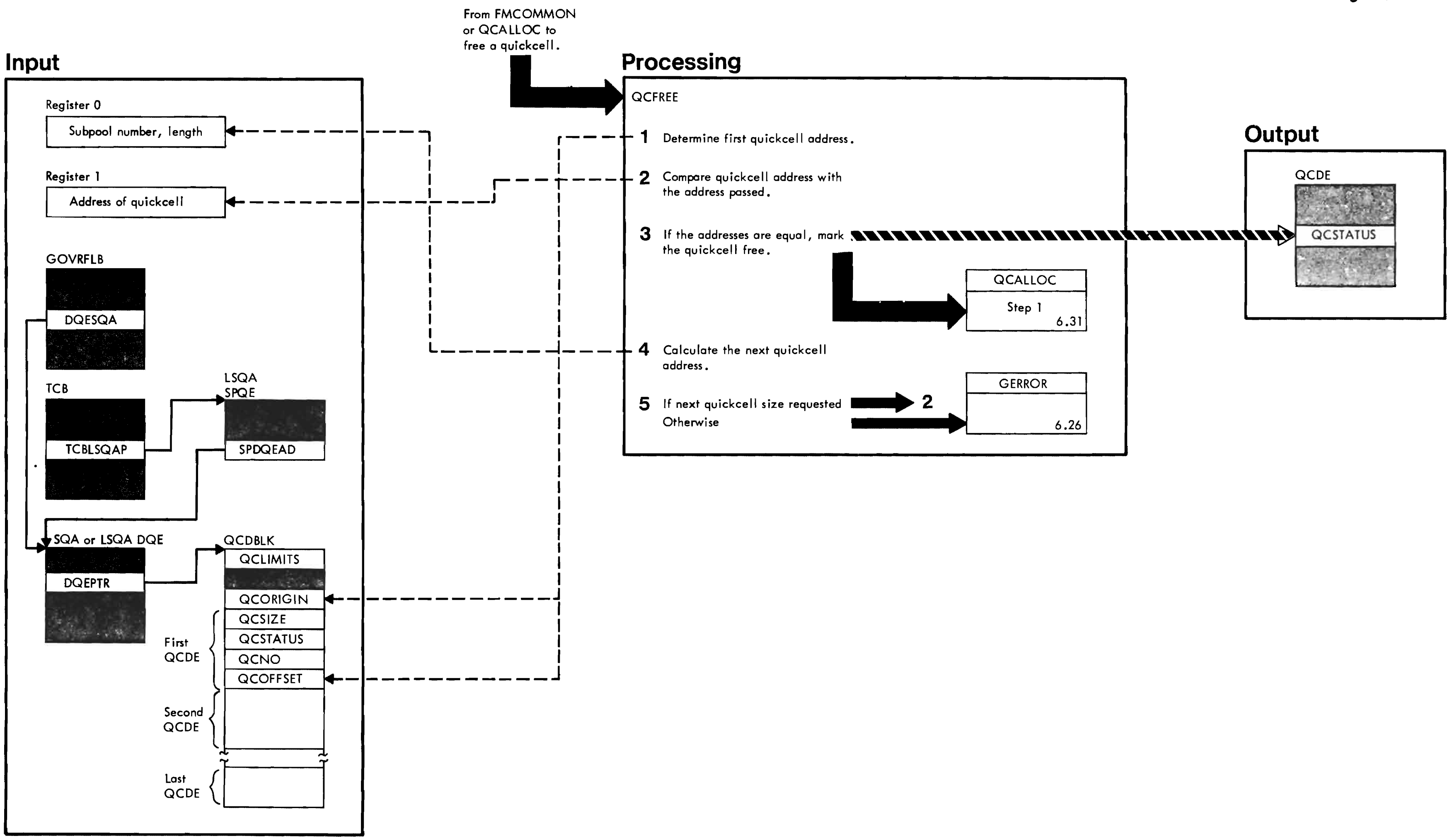
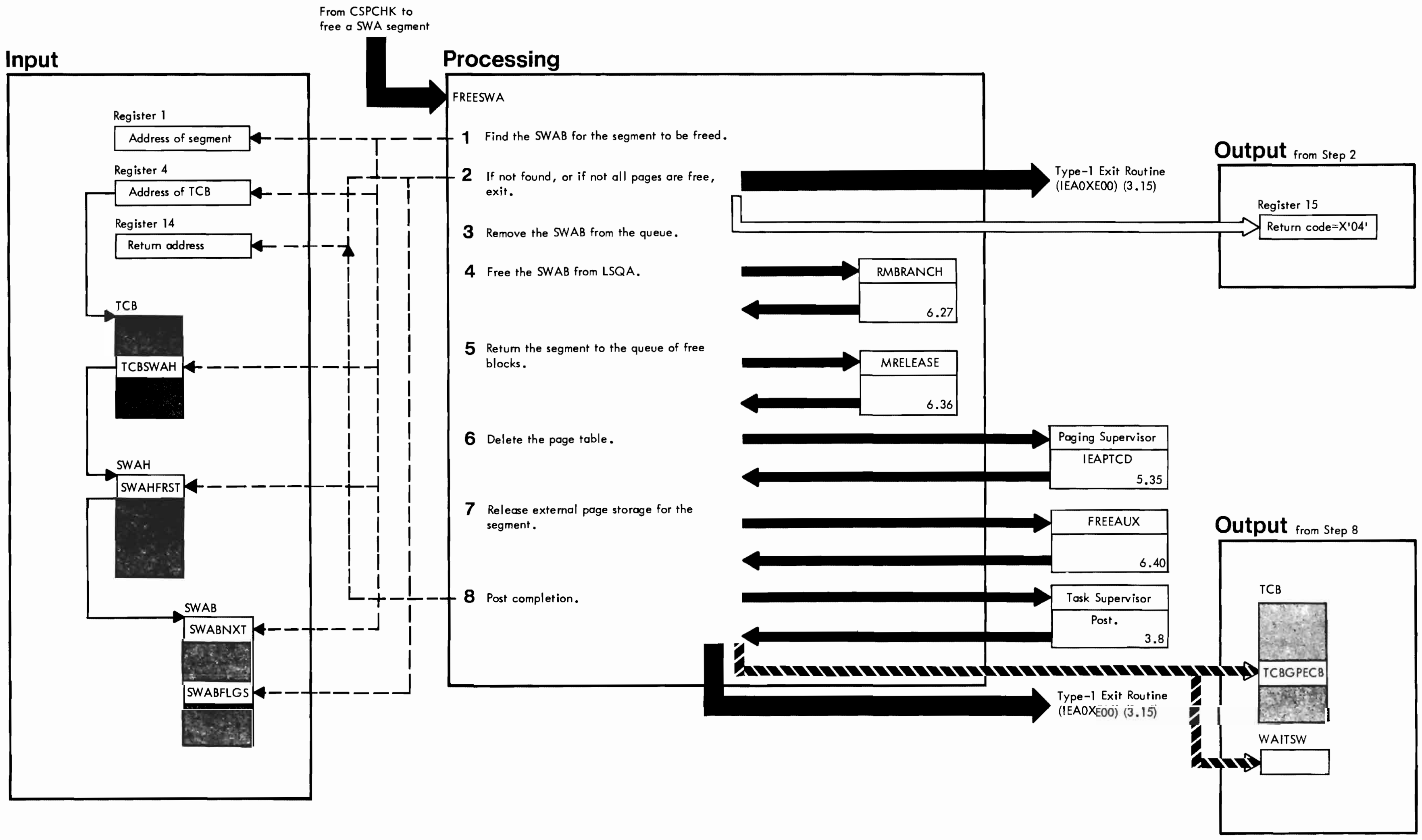

From FMCOMMON or QCALLOC to free a quickcell.
Input
Register 0
Subpool number, length
Register 1
Address of quickcell
GOVRFLB
DQESQA
TCB
TCBLSQAP
LSQA SPQE
SPDQEAD
SQA or LSQA DQE
DQEPTR
QCDBLK
QCLIMITS
QCORIGIN
QCSIZE
QCSTATUS
QCNO
QCOFFSET
First QCDE
Second QCDE
Last QCDE
Processing
QCFREE
1 Determine first quickcell address.
2 Compare quickcell address with the address passed.
3 If the addresses are equal, mark the quickcell free.
QCALLOC
Step 1
6.31
4 Calculate the next quickcell address.
5 If next quickcell size requested
Otherwise
2
GERROR
6.26
Output
QCDE
QCSTATUS

Diagram 6.37 Freeing a Quickcell (Module IEAVGM00)

NOTES	ROUTINE NAME	LABEL
1 The routine determines the address of the quickcell by adding QCORIGIN and QCOFFSET.	QCFREE	
2	QCFREEF	QRELEASE
3	QCFREE	QCHECKOK
4 The next quickcell address equals the previous address plus the length of the quickcell.		
5 Each time a quickcell of the specified length is checked, a counter is decremented. When the counter equals zero, all of them have been checked.		QCNONE

Diagram 6.38 Freeing a SWA Segment



FREESWA

1 Find the SWAB for the segment to be freed.

2 If not found, or if not all pages are free, exit.

3 Remove the SWAB from the queue.

4 Free the SWAB from LSQA.

5 Return the segment to the queue of free blocks.

6 Delete the page table.

7 Release external page storage for the segment.

8 Post completion.

Diagram 6.39
Freeing a SWA Page

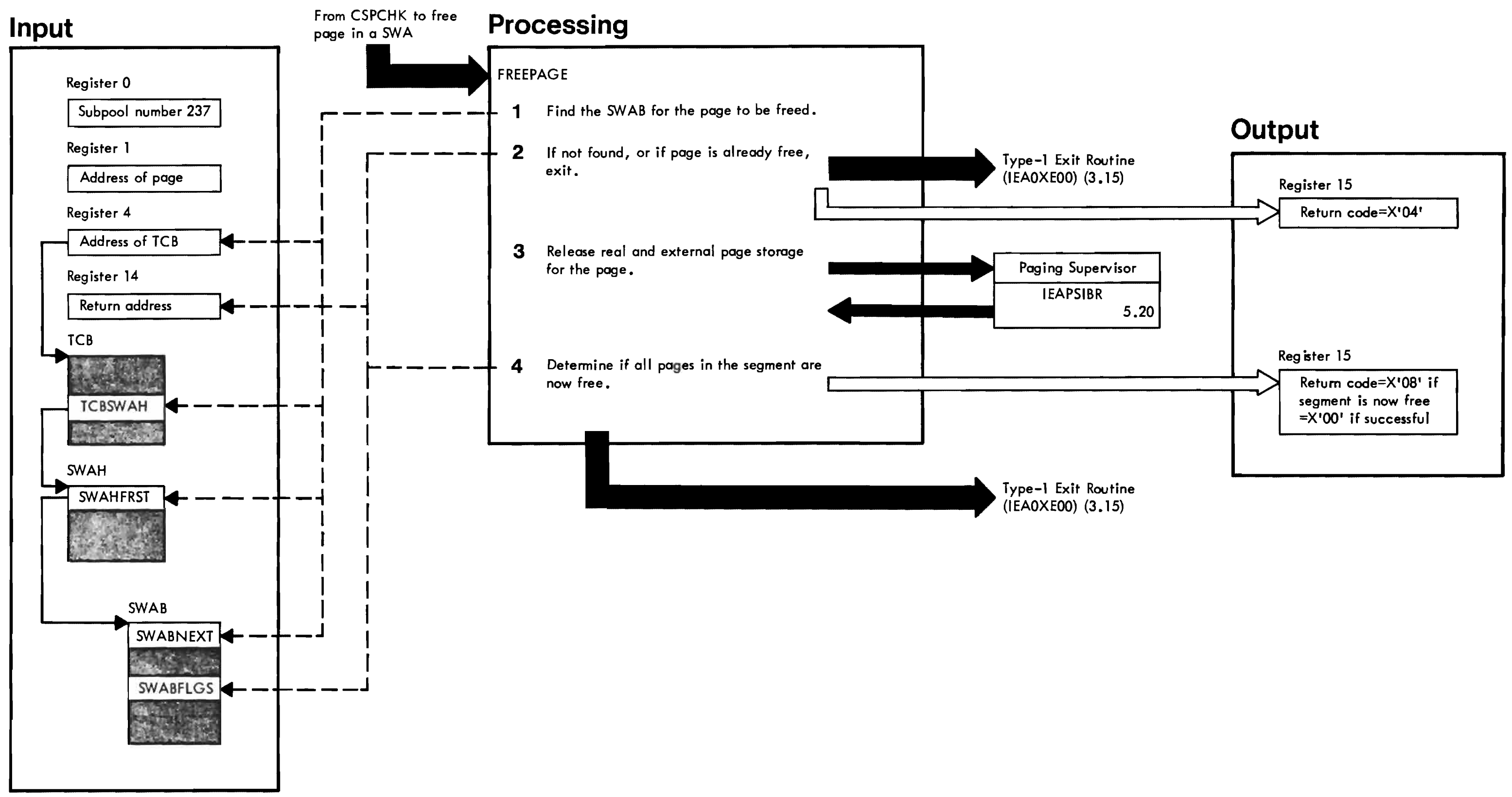

Input
Register 0
Subpool number 237
Register 1
Address of page
Register 4
Address of TCB
Register 14
Return address
TCB
TCBSWAH
SWAH
SWAHFRST
SWAB
SWABNEXT
SWABFLGS
From CSPCHK to free page in a SWA
Processing
FREEPAGE
1 Find the SWAB for the page to be freed.
2 If not found, or if page is already free, exit.
3 Release real and external page storage for the page.
4 Determine if all pages in the segment are now free.
Type-1 Exit Routine (IEAOXE00) (3.15)
Paging Supervisor
IEAPSIBR
5.20
Type-1 Exit Routine (IEAOXE00) (3.15)
Output
Register 15
Return code=X'04'
Register 15
Return code=X'08' if segment is now free =X'00' if successful

Diagram 6.40 Monitoring the Release of External Pages

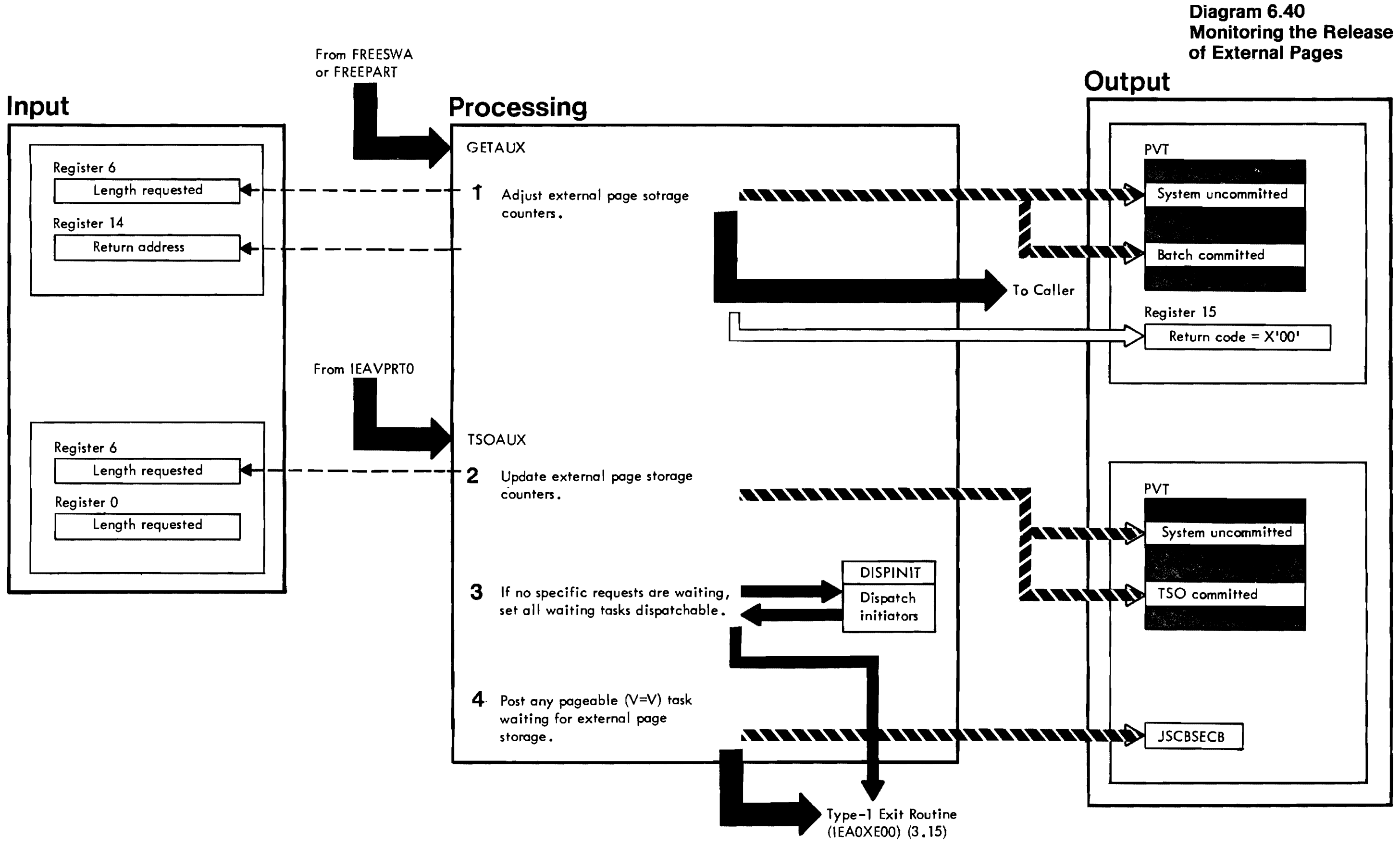


Diagram 6.40 Monitoring the Release of External Pages (Module IEAVPRTO)

NOTES	ROUTINE NAME	LABEL
The GETAUX subroutine is used to keep track of external page storage that backs up pageable regions for batch jobs. The TSOAUX subroutine is used to keep track of external page storage for TSO tasks.		
1 The system uncommitted count is increased and the batch committed count is decreased by the number of bytes requested.	GETAUX	
2 The system uncommitted count is increased and the TSO committed count is decreased by the number of bytes requested.	TSOAUX	

SECTION 7

Timer Supervision

7

The timer supervision routines support the System/370 time-of-day clock, clock comparator, and CPU timer. The routines use these to obtain the date and time of day, schedule activity after a specified interval, and schedule activity for a specific time of day.

The TOD (time-of-day) clock is a 64-bit incrementing binary counter. It is used to measure elapsed time in microseconds and is incremented by adding a one in bit-position 51 every microsecond. The clock, which runs continuously, contains the total number of microseconds that have elapsed since January 1, 1900, 00:00 hours, Greenwich mean time. Timer supervision uses the time-of-day clock to calculate the time of day and to maintain the current date.

The clock comparator feature causes an external interruption when the time-of-day clock reaches or passes the value specified in the clock comparator and the CPU is enabled for clock comparator interruptions. The interruption code is X'1004'. Timer supervision uses the comparator to service requests to schedule activity after a specified interval of real or wait time, or to schedule activity for a specific time of day.

The CPU timer is a 64-bit decrementing counter. One is subtracted from bit-position 51 every microsecond while the CPU is executing instructions or is in a wait state. An external interruption occurs when the value placed in the CPU timer reaches or becomes less than zero and the CPU is enabled for CPU timer interruptions. The interruption code is X'1005'. Timer supervision uses the CPU timer for timing task intervals.

Timer supervision performs the operations requested by TIME, STIMER, and TTIMER macro instructions, and processes timer interruptions.

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 programmed timer (the clock comparator or the CPU timer) to a requested time interval so that it expires after the specified time interval or at a specified time of day. When the requested interval expires, the CPU generates a CPU timer or clock comparator interruption which the Timer Second-Level Interruption Handler processes. If the requester specifies the TASK timing option, the time interval is decremented only when the requester's task is active. If the requester specifies the WAIT timing option, the requester's task is placed in the wait condition until the interval expires. If the register specifies the REAL timing option, the interval is decremented continuously.

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

Timer supervision maintains two queues of TQEs (timer queue elements): one for TASK timing requests and the other for REAL and WAIT timing requests. The TQEs are constructed by the STIMER routine, and each element represents a request for a type of 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 element from the appropriate timer queue and determines what action to take.

Timer supervision also maintains a timer data area (IEATPC). This area contains: (1) predefined TQEs that represent permanent control program requests for timer services, and (2) information used during timer supervision processing.

The four major routines and two queues in timer supervision are described briefly in the following paragraphs.

TIME ROUTINE

(Diagram 7.2)

The TIME routine supplies the current date and time of day. Initially, the operator uses the SET command to provide the control program with the date and the time of day at the computing location (local time of day). Timer supervision routines change the date at midnight and keep track of elapsed time. The TIME routine obtains the current local date, converts the value in the time-of-day clock to local time, and returns both values to the user.

STIMER ROUTINE

(Diagram 7.3)

The STIMER routine processes requests for interval timing based on task execution time or real time. The routine schedules the placement of a timing interval into either the clock comaprator or CPU timer, according to the timer service requested in the STIMER macro instruction. For each STIMER macro instruction, this routine builds or uses an existing TQE into which it places a summary of the information contained in the STIMER macro, including the information that will be needed when the interval expires. This routine then places the element on either the CPU timer queue (for the TASK timing option) or the clock comparator queue (for the REAL or WAIT timing option).

TTIMER ROUTINE

(Diagram 7.4)

The TTIMER routine supplies the time remaining in a previously requested interval, cancels previous timing requests, or both. To determine remaining time, the TTIMER routine subtracts elapsed time from the time at which the interval will expire. To cancel a previous request, it removes the corresponding element from either of the two timer queues.

TIMER SECOND-LEVEL INTERRUPTION HANDLER

(Diagram 7.5)

The Time Second-Level Interruption Handler (Timer SLIH), processes timer interruptions. It removes the TQE, whose interval has completed, from a timer queue and flags the interval complete. It performs any actions required upon expiration of the interval and obtains new intervals to be placed in the clock comparator. In addition, the Timer SLIH schedules or performs the following services: SMF timing, CVTDATE update, time-limit expiration, TSO timing, and paging supervision timing.

TIMER QUEUES

As mentioned previously, the timer supervision routines maintain two queues of timer queue elements: one for TASK timing requests and the other for REAL and WAIT timing requests.

Whenever a task requests a timer service, the STIMER routine builds a TQE. If the request is for TASK timing, the dispatcher requests that a timer supervision routine (Timer Enqueue) place the new TQE, and other related TQEs, on the CPU timer queue when the requesting task is dispatched. The timing interval requested by the dispatched task is placed in the CPU timer and decremented. If the task becomes nondispatchable, the remaining interval is saved. When the interval expires, the CPU timer causes an interruption. (See Figure 7-1.)

When a task requests REAL or WAIT timing, the STIMER routine converts the requested interval to the value that the time-of-day clock will contain when the requested interval expires. The TQE is placed on the clock comparator queue according to its time of expiration. The first TQE to expire is first on the queue, the second is second, and so on. The interval for the first TQE on the queue is converted to the value that the TOD clock will contain when the requested interval expires. This value is placed in the clock comparator, which causes an interruption when the value in the TOD clock equals or exceeds the value in the clock comparator. (See Figure 7-2.)

(1) Timer supervision routines maintain a pointer to the head of the CPU timer queue in the timer data area (IEATPC).

(2) When a task is dispatched, the dispatcher passes to the Timer Enqueue routine the addresses of all TQEs related to the task about to be dispatched. The Timer Enqueue routine places the TQE(s) on the CPU timer queue. If it places a TQE at the head of the CPU timer queue, it places that TQE's requested interval in the CPU timer. If a TQE is not placed at the head of the queue, the Timer Enqueue routine adjusts that TQE's requested interval to account for the intervals contained in any TQEs ahead of it on the queue. A given task can request only one TASK timing interval at a time. However, the control program requests timing of a user task in order to provide services for the entire system. These TQEs are "related" to the user TQE. The dispatcher times tasks for automatic priority grouping and time-slicing, the initiator for job-step timing, and so on.

(3) The dummy TQE is an internal TQE predefined in the timer data area. It is constructed to time the maximum allowable interval. When no other TQEs are on the queue, the value in the TQEVAL field of this TQE is placed in the CPU timer to prevent unexpected CPU timer interruptions.

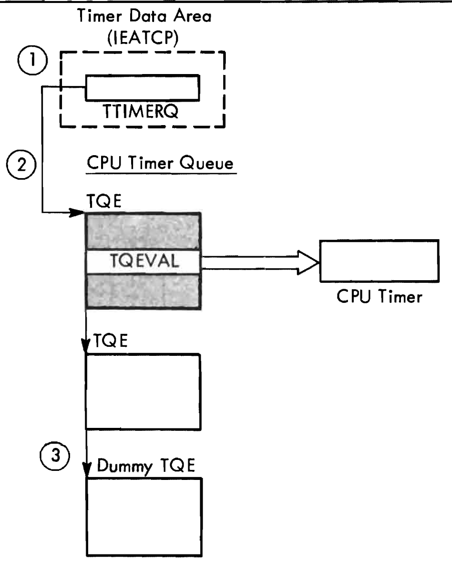


Figure 7-1. Operation of the task timing queue (for TASK timing).

(1) Timer supervision routines maintain a pointer to the head of the clock comparator queue in the timer data area (IEATPC).

(2) The STIMER routine calculates the value that the time-of-day clock will contain when the user's requested interval expires. This value is placed in the TQEVAL field of the requester's TQE. The TQEs are placed on the clock comparator queue according to their time of expiration. The TQE whose interval expires first is first on the queue, and so on. The expiration time (TQEVAL) of the first TQE on the queue is placed in the clock comparator.

(3) The paging supervision TQE (PGSUPTQE) is an internal TQE predefined in the timer data area. When the interval it represents expires, paging supervision gathers paging statistics.

(4) The TSO Driver TQE (IEATSELM) is an internal TQE predefined in the timer data area. It is used by the TSO Driver for time-slicing.

(5) The 10-minute SMF TQE (TENMELM) is an internal TQE predefined in the timer data area. This 10-minute element is used to cause an interruption every ten minutes for SMF wait-time collection. When the interval expires, the Timer SLIH adds the wait time accumulated to the total wait time kept in the system management control area.

(6) The Midnight TQE (MIDNTQE) is an internal TQE predefined in the timer data area. This TQE is used to determine when midnight occurs so the date in the CVT can be updated. The TQE is also used by the TIME routine to calculate the time of day.

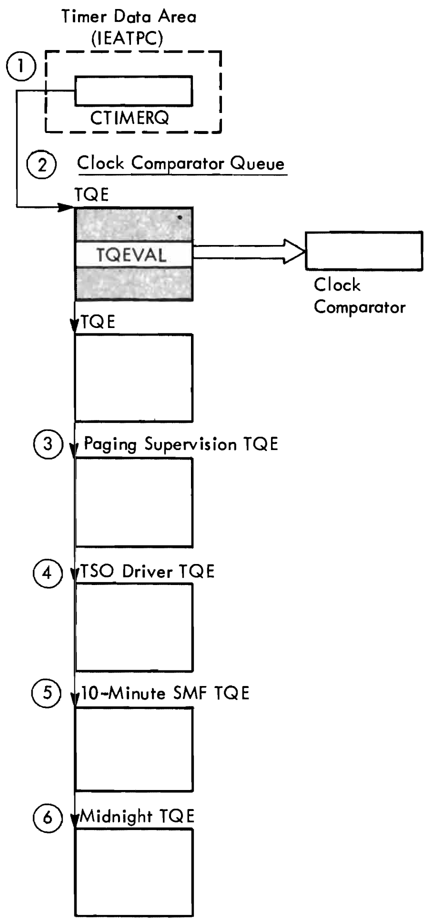


Figure 7-2. Operation of the clock comparator queue (for REAL and WAIT timing).

• Diagram 7.0 Timer Supervision Visual Table of Contents

Index to Diagrams by Module Name for Timer Supervision

Module Name	Diagram Number(s)
IEAVRT01	7.2
IEAVST00	7.3, 7.4
IEAVTI00	7.5--7.7

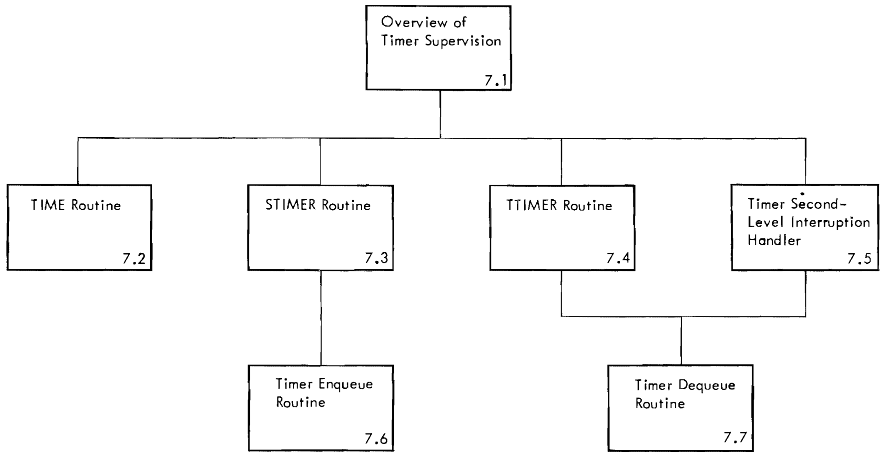
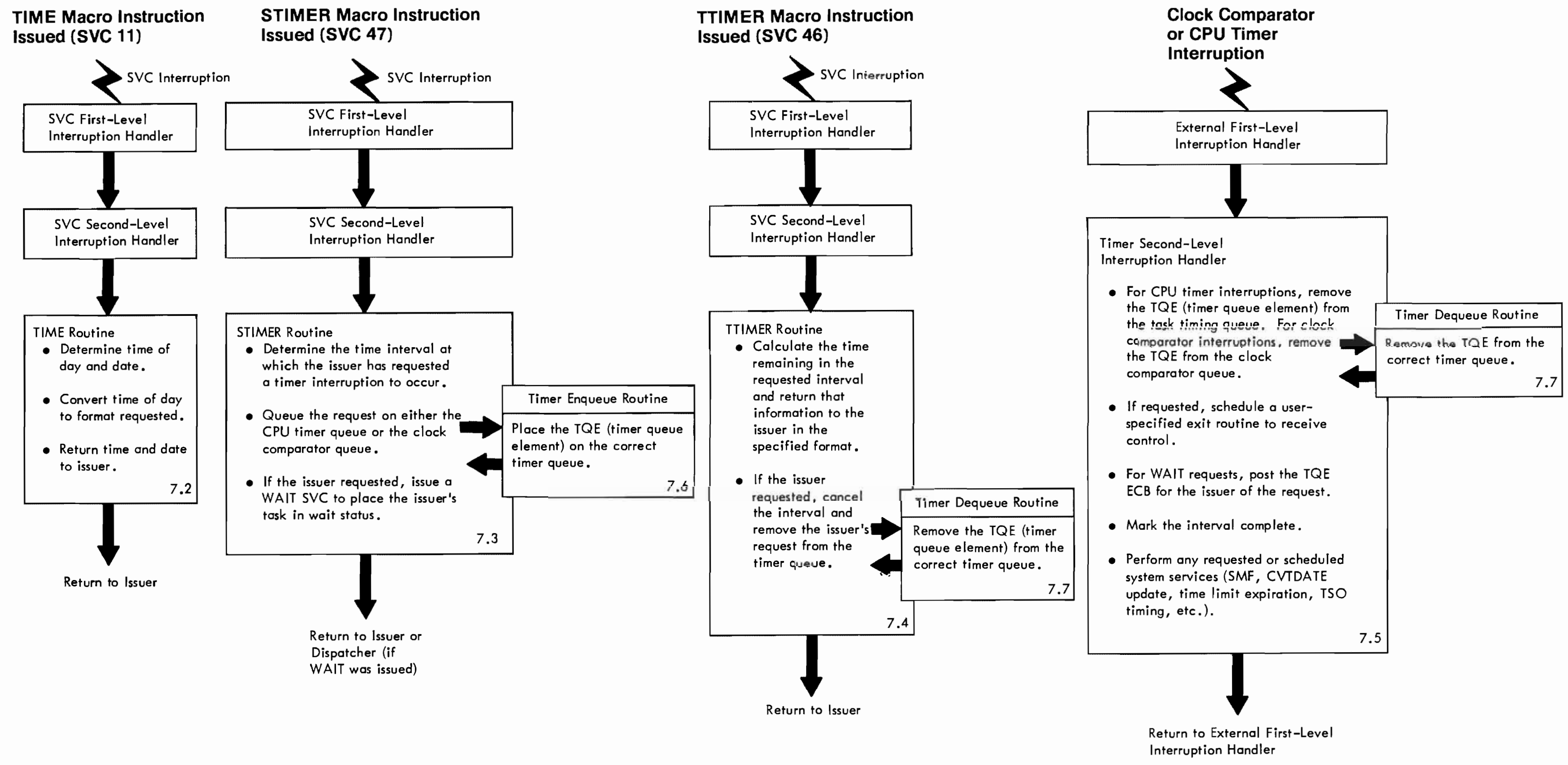

Overview of Timer Supervision 7.1
TIME Routine 7.2
STIMER Routine 7.3
TTIMER Routine 7.4
Timer Second-Level Interruption Handler 7.5
Timer Enqueue Routine 7.6
Timer Dequeue Routine 7.7

Diagram 7.1
Overview of Timer Supervision



TIME Macro Instruction Issued (SVC 11)
SVC Interruption
SVC First-Level Interruption Handler
SVC Second-Level Interruption Handler
TIME Routine
• Determine time of day and date.
• Convert time of day to format requested.
• Return time and date to issuer.
7.2
Return to Issuer
STIMER Macro Instruction Issued (SVC 47)
SVC Interruption
SVC First-Level Interruption Handler
SVC Second-Level Interruption Handler
STIMER Routine
• Determine the time interval at which the issuer has requested a timer interruption to occur.
• Queue the request on either the CPU timer queue or the clock comparator queue.
• If the issuer requested, issue a WAIT SVC to place the issuer's task in wait status.
7.3
Timer Enqueue Routine
Place the TQE (timer queue element) on the correct timer queue.
7.6
Return to Issuer or Dispatcher (if WAIT was issued)
TTIMER Macro Instruction Issued (SVC 46)
SVC Interruption
SVC First-Level Interruption Handler
SVC Second-Level Interruption Handler
TTIMER Routine
• Calculate the time remaining in the requested interval and return that information to the issuer in the specified format.
• If the issuer requested, cancel the interval and remove the issuer's request from the timer queue.
7.4
Timer Dequeue Routine
Remove the TQE (timer queue element) from the correct timer queue.
7.7
Return to Issuer
Clock Comparator or CPU Timer Interruption
External First-Level Interruption Handler
Timer Second-Level Interruption Handler
• For CPU timer interruptions, remove the TQE (timer queue element) from the task timing queue. For clock comparator interruptions, remove the TQE from the clock comparator queue.
• If requested, schedule a user-specified exit routine to receive control.
• For WAIT requests, post the TQE ECB for the issuer of the request.
• Mark the interval complete.
• Perform any requested or scheduled system services (SMF, CVTDATE update, time limit expiration, TSO timing, etc.).
7.5
Timer Dequeue Routine
Remove the TQE from the correct timer queue.
7.7
Return to External First-Level Interruption Handler

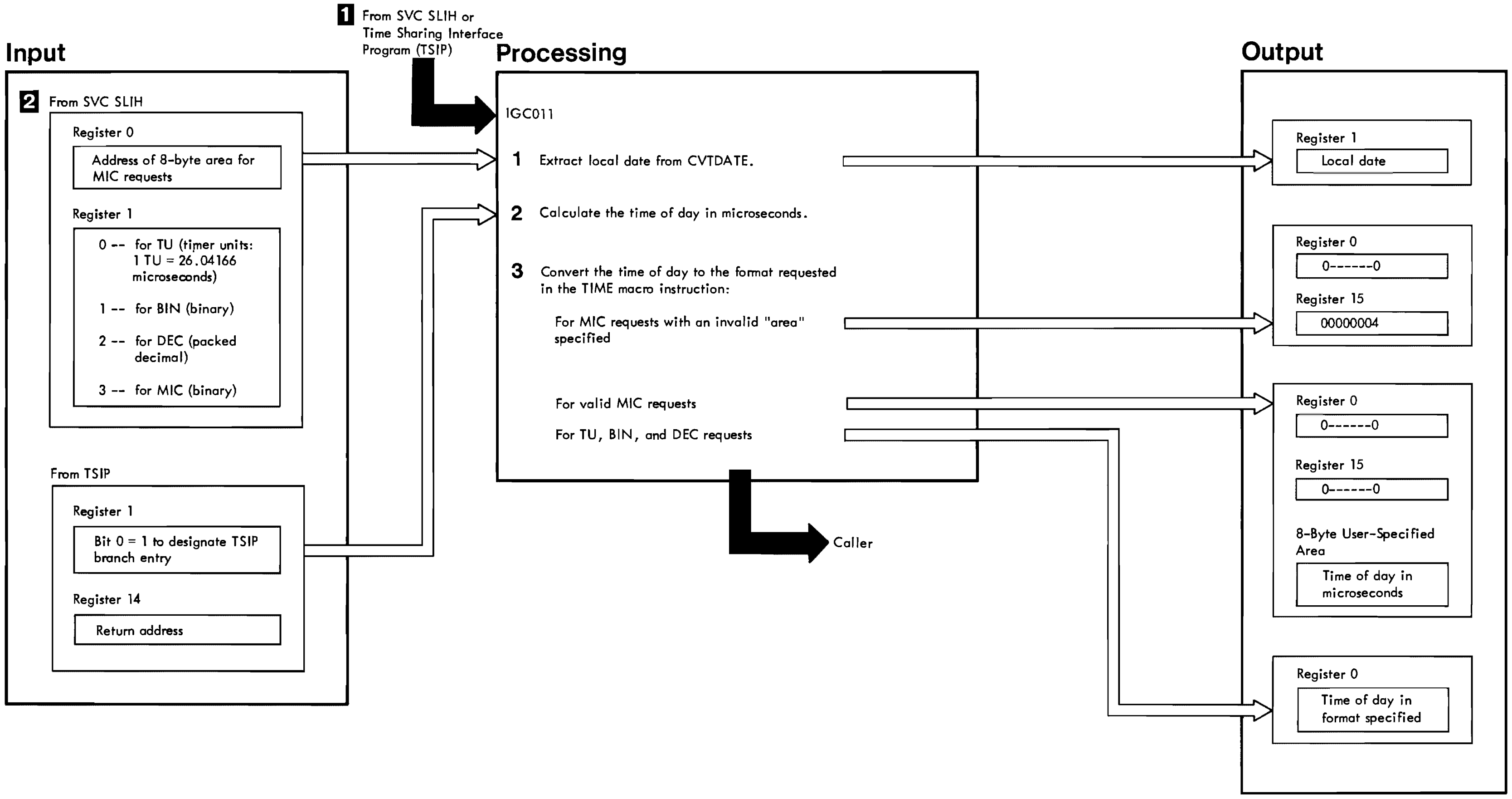
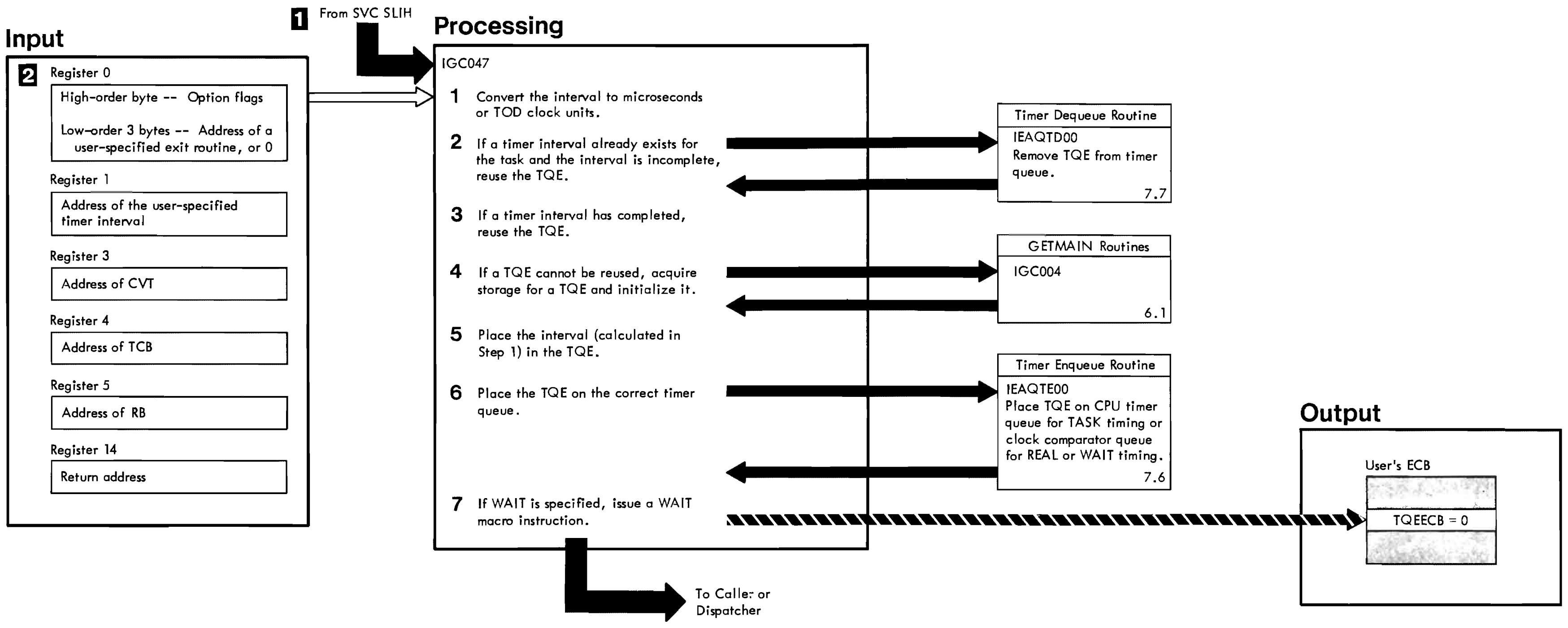

Input
2 From SVC SLIH
Register 0
Address of 8-byte area for MIC requests
Register 1
0 -- for TU (timer units: 1 TU = 26.04166 microseconds)
1 -- for BIN (binary)
2 -- for DEC (packed decimal)
3 -- for MIC (binary)
From TSIP
Register 1
Bit 0 = 1 to designate TSIP branch entry
Register 14
Return address
1 From SVC SLIH or Time Sharing Interface Program (TSIP)
Processing
IGC011
1 Extract local date from CVTDATE.
2 Calculate the time of day in microseconds.
3 Convert the time of day to the format requested in the TIME macro instruction:
For MIC requests with an invalid "area" specified
For valid MIC requests
For TU, BIN, and DEC requests
Caller
Output
Register 1
Local date
Register 0
0------0
Register 15
00000004
Register 0
0------0
Register 15
0------0
8-Byte User-Specified Area
Time of day in microseconds
Register 0
Time of day in format specified

Diagram 7.2 TIME Routine (Module IEAVRT01)

NOTES	ROUTINE NAME	LABEL
$\boxed{1}$ The TIME routine services TIME macro instructions. It retrieves the local date, calculates the local time of day, and returns both to the requester specified in the macro instruction. The routine is a type-2, partially-enabled SVC routine. It is entered from the SVC Second-Level Interruption Handler (SVC SLIH) after an SVC 11 (TIME SVC) is issued, or from the Time Sharing Interface Program (TSIP) by a branch entry) -- to obtain the time of day in timer units.	TIME	
$\boxed{2}$ Register 0 contains the address of an 8-byte area in which the time of day is placed if the MICVI option is specified by the requester. Register 1 specifies the format in which the time of day is to be returned to the requester.		
1 If the TIME routine is entered prior to system initialization, the date has not yet been set (CVTDATE=0 or MNIGHT=0). Register 0 is set to zero, register 1 is set to X'F', and the TIME routine returns.		TDATE
2 The time-of-day (TOD) clock contains a count of the total number of microseconds that have elapsed since January 1, 1900, Greenwich mean time (GMT). When a request for the time of day is made, this elapsed time must be converted to the number of microseconds that have elapsed in the day at the local computing center (local time). To convert the TOD clock value to local time, the TIME routine maintains a field (MNIGHT) in the midnight TQE, which contains the value that the TOD clock will contain when midnight occurs (local time). This field is initially set when the operator issues the SET command. The following method is used to calculate the local time of day: MNIGHT field (number of microseconds TOD clock will contain at local midnight) - TOD clock (number of microseconds elapsed since January 1, 1900, GMT) --- MICSEC value (number of microseconds left in the day at the computing location) Then: 86.4×10^9 (number of microseconds in 24 hours) - MICSEC value --- Local time of day at the computing location in microseconds.		

Diagram 7.3 STIMER Routine



1 From SVC SLIH

Input

2

Register 0

High-order byte -- Option flags

Low-order 3 bytes -- Address of a user-specified exit routine, or 0

Register 1

Address of the user-specified timer interval

Register 3

Address of CVT

Register 4

Address of TCB

Register 5

Address of RB

Register 14

Return address

Processing

IGC047

1 Convert the interval to microseconds or TOD clock units.

2 If a timer interval already exists for the task and the interval is incomplete, reuse the TQE.

3 If a timer interval has completed, reuse the TQE.

4 If a TQE cannot be reused, acquire storage for a TQE and initialize it.

5 Place the interval (calculated in Step 1) in the TQE.

6 Place the TQE on the correct timer queue.

7 If WAIT is specified, issue a WAIT macro instruction.

Timer Dequeue Routine
IEAQTD00
Remove TQE from timer queue.
7.7

GETMAIN Routines
IGC004
6.1

Timer Enqueue Routine
IEAQTE00
Place TQE on CPU timer queue for TASK timing or clock comparator queue for REAL or WAIT timing.
7.6

To Caller or Dispatcher

Output

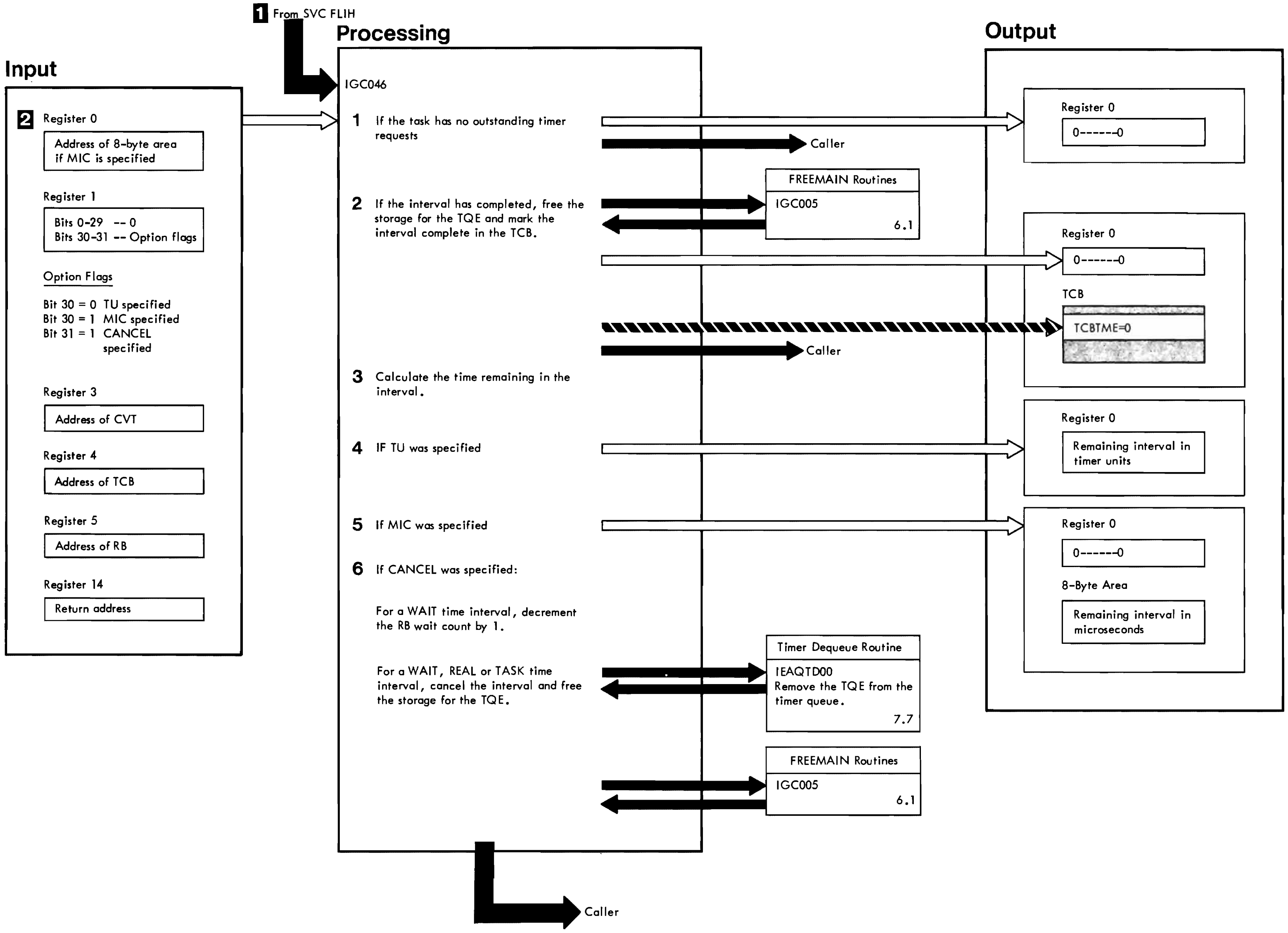
User's ECB

TQEECB = 0

Diagram 7.3 STIMER Routine (Module IEAVST00)

NOTES	ROUTINE NAME	LABEL
1 The STIMER routine processes STIMER macro instructions. The routine sets a programmed timer to expire either: (1) after a specified time interval, or (2) at a specified time of day. The task requesting the timed interval is placed in wait state during the interval if the task so specifies. The STIMER routine also prepares to pass control to a user-specified exit routine upon expiration of the interval if requested to do so. <u>Restrictions</u>: • If an STIMER macro instruction is issued by a user-specified exit routine, the macro instruction will be executed. However, the macro instruction must not specify the same exit routine. If it does, an infinite loop may occur. • The specified time interval should be less than 24 hours. If greater, the interval is reduced to 24 hours. • After an interval expires, the task is placed in the ready state and competes for control of the CPU. A user-specified exit routine will not receive control until the task becomes the highest priority ready task and is dispatched. • A task can have only one outstanding STIMER service request. A second STIMER macro instruction issued for the same task overrides the first.	STIMER	

NOTES	ROUTINE NAME	LABEL
2 Register 0 / STIMER Macro <u>Option Flags</u> / <u>Instruction Option</u> .000 .... TUINTVL indicates the .001 .... BINTVL format .010 .... MICVL of specified .011 .... DINTVL time interval .111 .... LTOD .... .000 TASK .... .001 WAIT .... .011 REAL Register 1 <u>Option</u> / <u>Register Contents</u> DINTVL,MICVL,LTOD — The address of an 8-byte area containing the time value BINTVL, TUINTVL — The address of a 4-byte area containing the time value	STIMER	
2 If the task has an outstanding timer WAIT request the waiting RB is located. If the wait count (RBWCF) in the RB is not zero, it is decremented by one before processing continues.		TCBT
3		TIRBTST
4		TGETCORE
7 Issuing the WAIT macro instruction places the user's SVRB in the wait state. When the interval has expired, the Timer SLIH posts the TQEECB field.		



Input

2 Register 0

Address of 8-byte area if MIC is specified

Register 1

Bits 0-29 -- 0
Bits 30-31 -- Option flags

Option Flags

Bit 30 = 0	TU specified
Bit 30 = 1	MIC specified
Bit 31 = 1	CANCEL specified

Register 3

Address of CVT

Register 4

Address of TCB

Register 5

Address of RB

Register 14

Return address

Processing

1 From SVC FLIH

IGC046

1 If the task has no outstanding timer requests

2 If the interval has completed, free the storage for the TQE and mark the interval complete in the TCB.

3 Calculate the time remaining in the interval.

4 IF TU was specified

5 If MIC was specified

6 If CANCEL was specified:

For a WAIT time interval, decrement the RB wait count by 1.

For a WAIT, REAL or TASK time interval, cancel the interval and free the storage for the TQE.

Output

Register 0

0------0

Register 0

0------0

TCB

TCBTME=0

Register 0

Remaining interval in timer units

Register 0

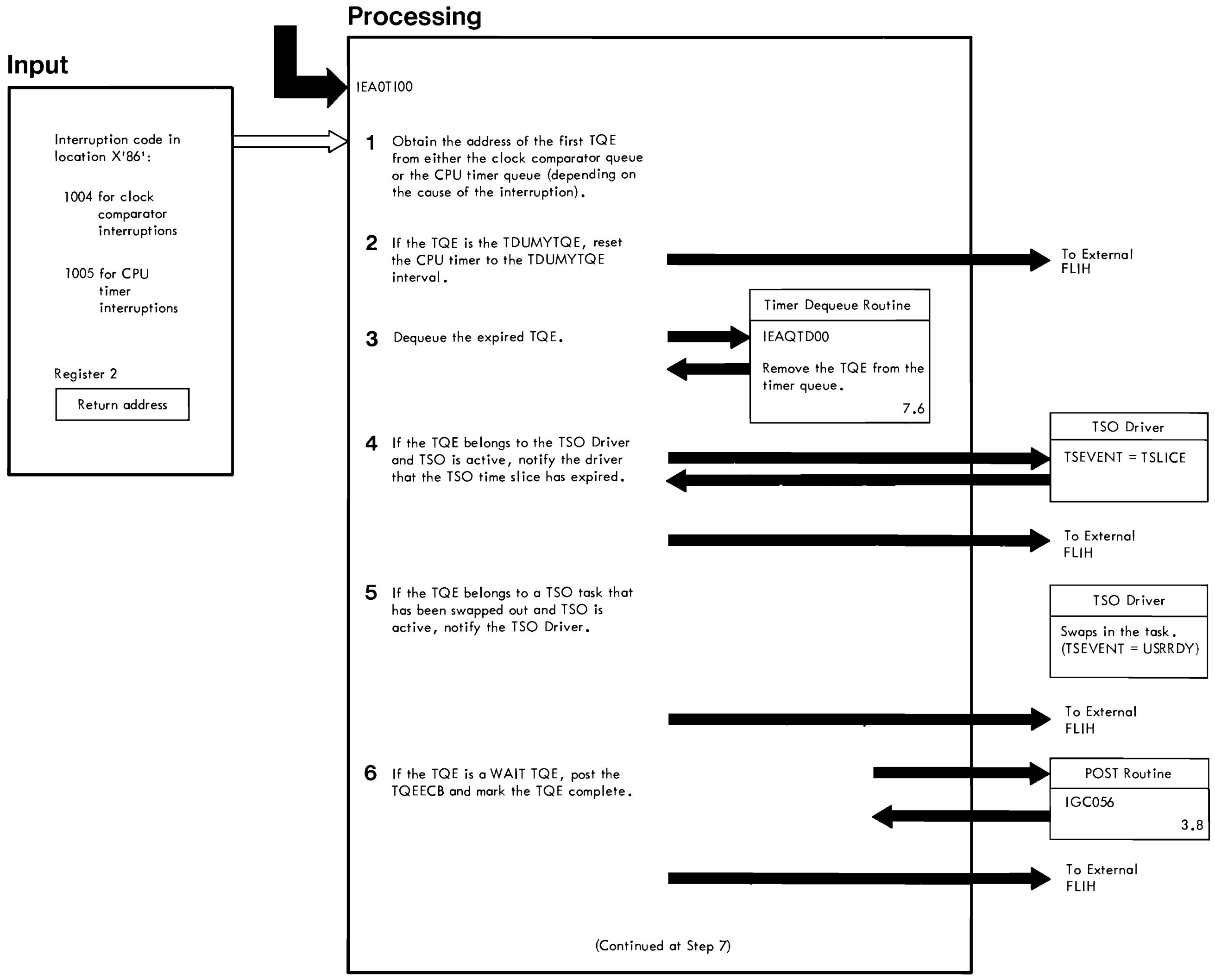
0------0

8-Byte Area

Remaining interval in microseconds

Diagram 7.4 TTIMER Routine (Module IEAVST00)

NOTES	ROUTINE NAME	LABEL
1 The TTIMER routine processes TTIMER macro instructions. The routine calculates the time remaining in a timer interval and optionally cancels that interval. The time remaining is returned to the requester as specified in the macro instruction. If the WAIT parameter was specified in the STIMER macro instruction that established the interval, the interval is also canceled. This routine is a type-2, partially-enabled SVC routine. It is entered from the SVC SLIH after an SVC 46 (TTIMER SVC) is issued.	TTIMER	
2 If the TUINTVL option is specified, the time remaining in the interval is to be returned in register 0 in timer units (the least significant bit = 26.04166 micro-seconds).		
If the MICVL parameter is specified, the time remaining in the interval is to be returned in the area whose address is specified in register 0. Bit 51 equals 1 microsecond.		
3 For TASK time intervals:		TSET2A
• If the TQE is first on the CPU timer queue, the remaining interval equals the value in the CPU timer.		
• If the TQE is not first on the queue, the remaining interval equals TQEVAL minus INTERVAL plus the value in the CPU timer. INTERVAL is a field in the timer data area. The field contains the amount of time that will have elapsed (when the CPU timer reaches zero) since the first TQE for the current task was put on the CPU timer queue.		
For WAIT and REAL time intervals:		TSET2
• The value in the TOD clock is placed in the IEACLOCK field. This value is subtracted from the TQEVAL field of the requester's TQE to determine the remaining interval.		



Input

Interruption code in location X'86':

1004 for clock comparator interruptions

1005 for CPU timer interruptions

Register 2

Return address

Processing

IEAOT100

1 Obtain the address of the first TQE from either the clock comparator queue or the CPU timer queue (depending on the cause of the interruption).

2 If the TQE is the TDUMYTQE, reset the CPU timer to the TDUMYTQE interval.

3 Dequeue the expired TQE.

4 If the TQE belongs to the TSO Driver and TSO is active, notify the driver that the TSO time slice has expired.

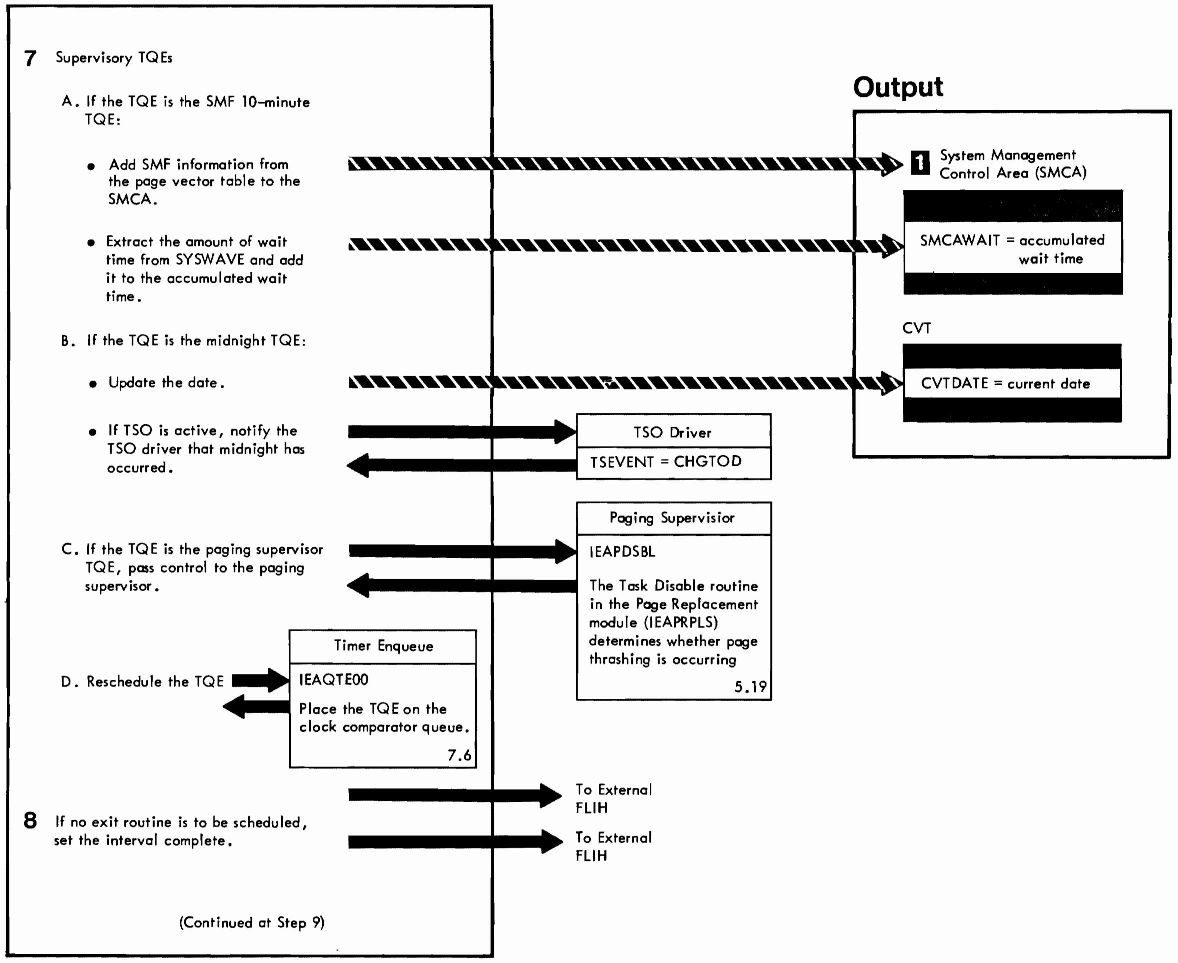
5 If the TQE belongs to a TSO task that has been swapped out and TSO is active, notify the TSO Driver.

6 If the TQE is a WAIT TQE, post the TQEECB and mark the TQE complete.

(Continued at Step 7)

Diagram 7.5 (Steps 1-6) Timer Second-Level Interruption Handler (Module IEAVTI00)

NOTES		ROUTINE NAME	LABEL
2	The dummy TQE is predefined in the timer data area (IEATPC). It contains a value greater than the maximum allowed user interval. The dummy TQE is scheduled when no other TQEs are on the queue in order to prevent unexpected CPU timer interruptions.	Timer SLIH	
3			TIDQTQE
4	If the TQE belongs to the TSO Driver but TSO is not active, the Timer SLIH exits to the External FLIH.		
5	The TSO Restore routine will complete processing of the TQE after the TSO task is swapped in.		
6			TIWAIT



Processing

7 Supervisory TQEs

A. If the TQE is the SMF 10-minute TQE:

- Add SMF information from the page vector table to the SMCA.
- Extract the amount of wait time from SYSWAVE and add it to the accumulated wait time.

B. If the TQE is the midnight TQE:

- Update the date.
- If TSO is active, notify the TSO driver that midnight has occurred.

C. If the TQE is the paging supervisor TQE, pass control to the paging supervisor.

D. Reschedule the TQE

8 If no exit routine is to be scheduled, set the interval complete.

(Continued at Step 9)

TSO Driver
TSEVENT = CHGTOD
Paging Supervisior
IEAPDSBL
The Task Disable routine in the Page Replacement module (IEAPRPLS) determines whether page thrashing is occurring
5.19
Timer Enqueue
IEAQTE00
Place the TQE on the clock comparator queue.
7.6
To External FLIH
To External FLIH
Output
1 System Management Control Area (SMCA)
SMCAWAIT = accumulated wait time
CVT
CVTDATE = current date

Diagram 7.5 (Steps 7-8) Timer Second-Level Interruption Handler (Module IEAVTI00)

NOTES	ROUTINE NAME	LABEL
7 A.	Timer SLIH	TISMFMID
C. The paging supervision TQE is used to determine how much paging occurs within a certain time period. The Task Disable routine will set a task nondispatchable if too much paging (thrashing) is being done.		
D. The value in the TOD clock is added to the TQEVAL field of the TQE and placed in TQEVAL.		TSNQUEUE
8 This is a user's TQE		NOTUSER
1 The Timer SLIH adds the following page vector table (PVT) fields to the SMF control area (SMCA):		

<u>PVT Field</u>	<u>SMF Field</u>	<u>Field Contents</u>
PVTNPOUT	SMCAPGOT	Number of page-outs
PVTNPIN	SMCAPGIN	Number of page-ins
PVTSPOUT	SMCASPCT	Number of SWAP page-outs
PVTSPIN	SMCASPIN	Number of SWAP page-ins
PVTNSWAP	SMCAPGNS	Number of regions swapped in and out
PVTNPMIG	SMCAPGM	Pages migrated
PVTNPEEC	SMCAPGEL	Number of pages reclaimed
PVTNRM	SMCAPGNM	Regions migrated

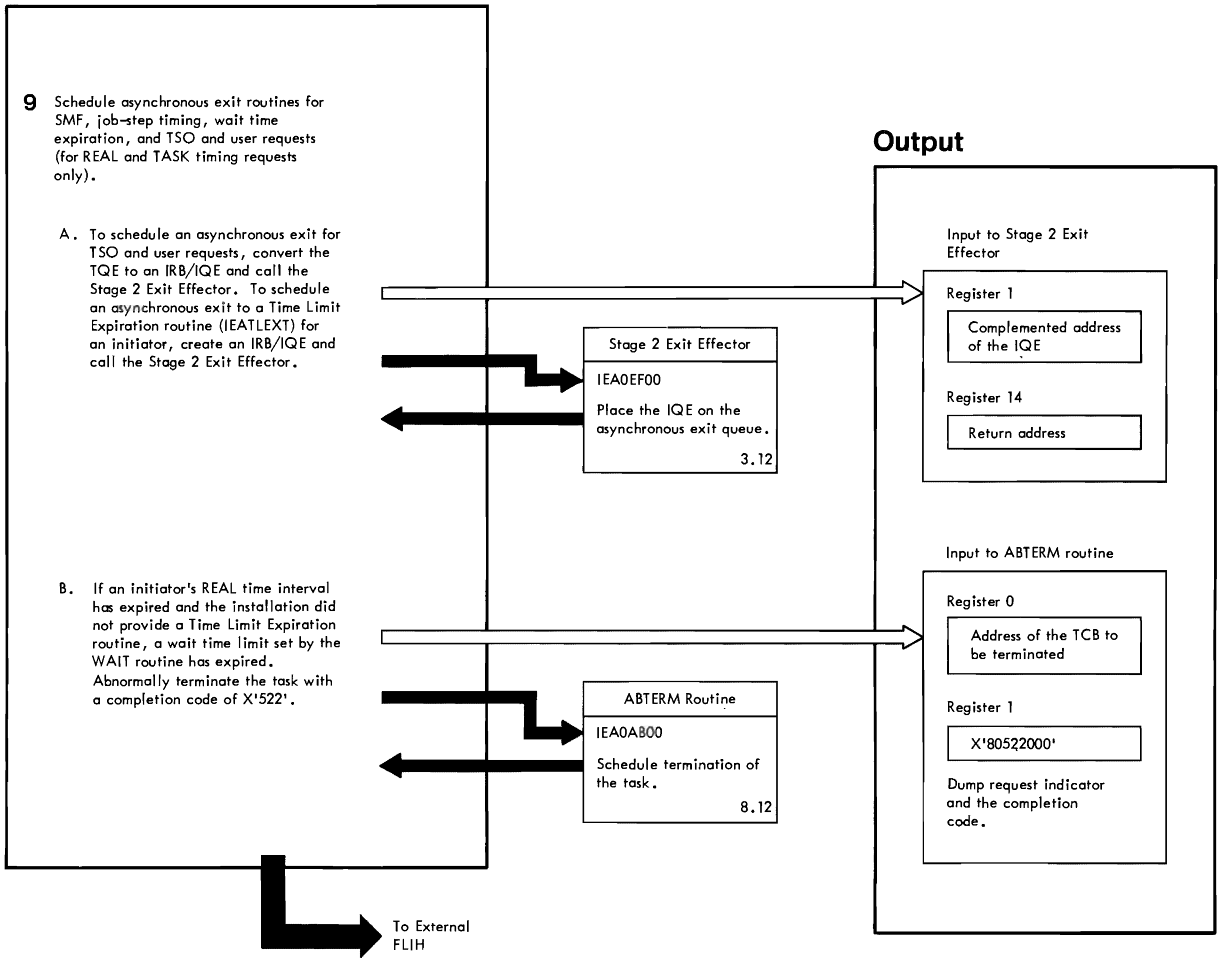
The Timer SLIH also places the time of day that the 10-minute TQE expired in the field SMCATEXP.

Processing

9 Schedule asynchronous exit routines for SMF, job-step timing, wait time expiration, and TSO and user requests (for REAL and TASK timing requests only).

A. To schedule an asynchronous exit for TSO and user requests, convert the TQE to an IRB/IQE and call the Stage 2 Exit Effector. To schedule an asynchronous exit to a Time Limit Expiration routine (IEATLEXT) for an initiator, create an IRB/IQE and call the Stage 2 Exit Effector.

B. If an initiator's REAL time interval has expired and the installation did not provide a Time Limit Expiration routine, a wait time limit set by the WAIT routine has expired. Abnormally terminate the task with a completion code of X'522'.

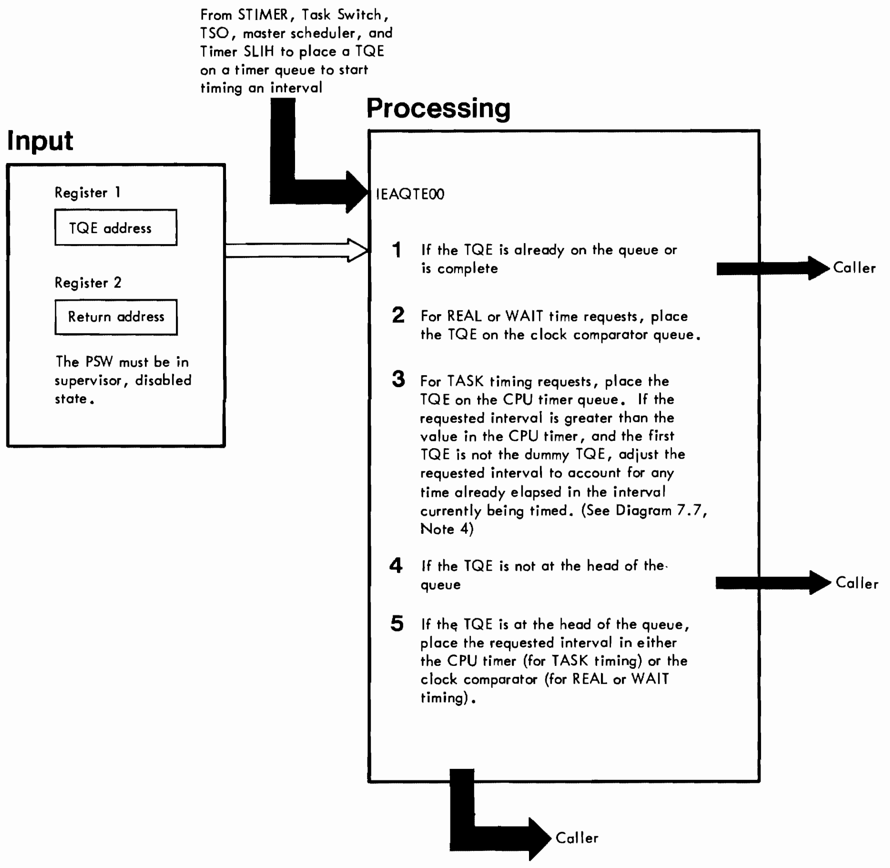

Stage 2 Exit Effector
IEAOEF00
Place the IQE on the asynchronous exit queue.
3.12
ABTERM Routine
IEAOAB00
Schedule termination of the task.
8.12
To External FLIH

Output

Input to Stage 2 Exit Effector
Register 1
Complemented address of the IQE
Register 14
Return address
Input to ABTERM routine
Register 0
Address of the TCB to be terminated
Register 1
X'80522000'
Dump request indicator and the completion code.

Diagram 7.5 (Step 9) Timer Second-Level Interruption Handler (Module IEAVTI00)

NOTES	ROUTINE NAME	LABEL
9 A.	Timer SLIH	TNJST CHECKSMF
B. Register 0 contains the address of the TCB to be terminated. This TCB is the subtask of the TCB to which the TQE points. Upon return from ABTERM, the Timer SLIH moves the TQESAV field to the TQEVAL field to restore the time interval in effect when the job step entered the wait state (for debugging purposes). The TQE type is then changed from REAL to TASK.		



From STIMER, Task Switch, TSO, master scheduler, and Timer SLIH to place a TQE on a timer queue to start timing an interval

Input

Register 1

TQE address

Register 2

Return address

The PSW must be in supervisor, disabled state.

Processing

IEAQTE00

1 If the TQE is already on the queue or is complete → Caller

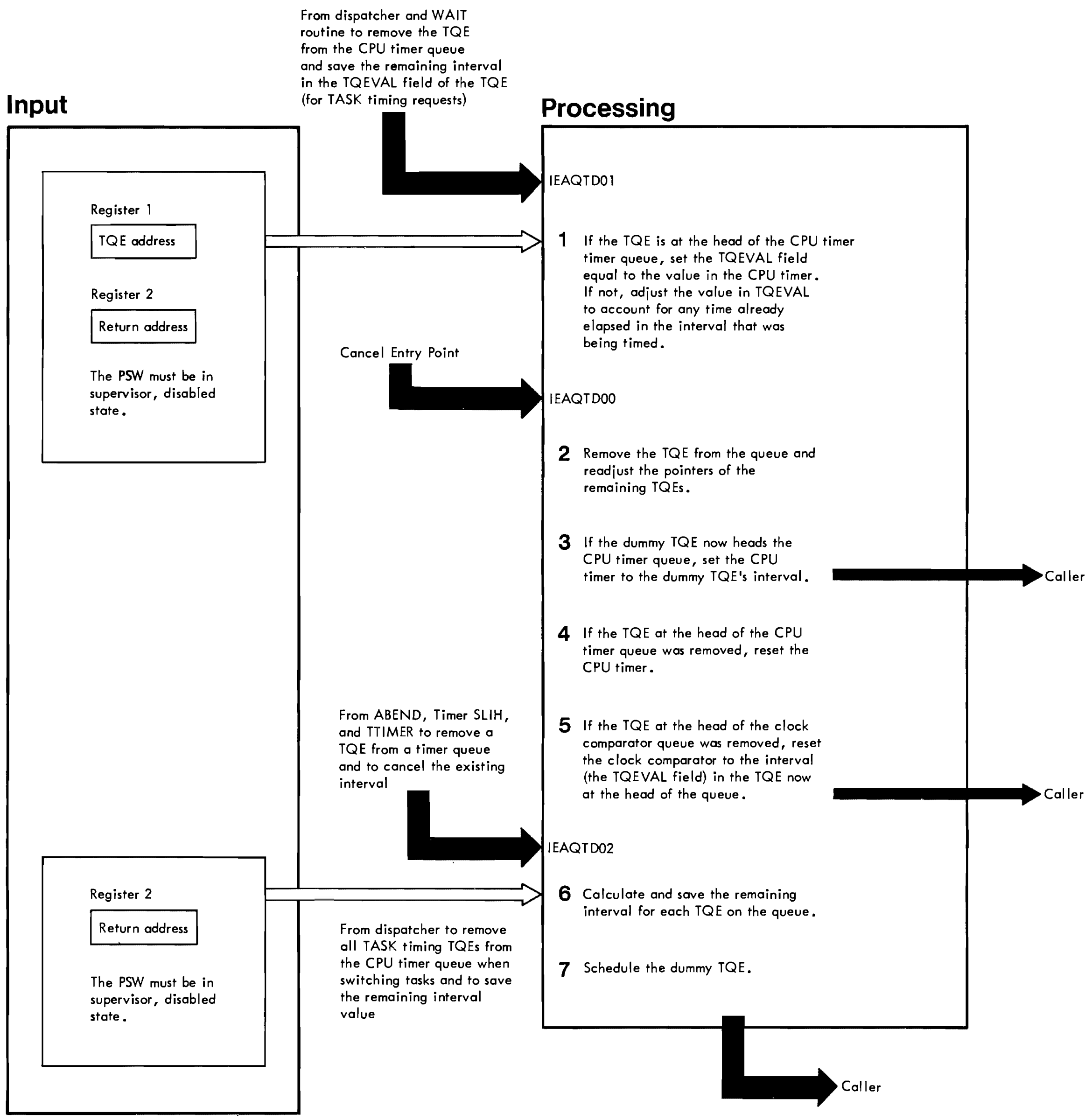
2 For REAL or WAIT time requests, place the TQE on the clock comparator queue.

3 For TASK timing requests, place the TQE on the CPU timer queue. If the requested interval is greater than the value in the CPU timer, and the first TQE is not the dummy TQE, adjust the requested interval to account for any time already elapsed in the interval currently being timed. (See Diagram 7.7, Note 4)

4 If the TQE is not at the head of the queue → Caller

5 If the TQE is at the head of the queue, place the requested interval in either the CPU timer (for TASK timing) or the clock comparator (for REAL or WAIT timing).

→ Caller


Input
Register 1
TQE address
Register 2
Return address
The PSW must be in supervisor, disabled state.
Register 2
Return address
The PSW must be in supervisor, disabled state.
From dispatcher and WAIT routine to remove the TQE from the CPU timer queue and save the remaining interval in the TQEVAL field of the TQE (for TASK timing requests)
Cancel Entry Point
From ABEND, Timer SLIH, and TTIMER to remove a TQE from a timer queue and to cancel the existing interval
From dispatcher to remove all TASK timing TQEs from the CPU timer queue when switching tasks and to save the remaining interval value
Processing
IEAQTD01
IEAQTD00
IEAQTD02
Caller
Caller
Caller

1 If the TQE is at the head of the CPU timer timer queue, set the TQEVAL field equal to the value in the CPU timer. If not, adjust the value in TQEVAL to account for any time already elapsed in the interval that was being timed.

2 Remove the TQE from the queue and readjust the pointers of the remaining TQEs.

3 If the dummy TQE now heads the CPU timer queue, set the CPU timer to the dummy TQE's interval.

4 If the TQE at the head of the CPU timer queue was removed, reset the CPU timer.

5 If the TQE at the head of the clock comparator queue was removed, reset the clock comparator to the interval (the TQEVAL field) in the TQE now at the head of the queue.

6 Calculate and save the remaining interval for each TQE on the queue.

7 Schedule the dummy TQE.

Diagram 7.7 Timer Dequeue Routine (Module IEAVTI00)

NOTES	ROUTINE NAME	LABEL
1 The TQEVAL field is set to: old TQEVAL - INTERVAL + the value in the CPU timer or 0 if the remaining time is negative where: "old TQEVAL" contains the requested interval INTERVAL is a field in the timer data area that contains the requested interval in the TQE now removed from the head of the queue. **4** The requested interval in the TQE must be adjusted to account for any time already elapsed in the interval that was being timed. The CPU timer is set to: TQEVAL - INTERVAL + the value in the CPU timer where: TQEVAL is the requested interval. INTERVAL is a field in the timer data area that contains the interval requested in the TQE now removed from the head of the queue.	Timer Dequeue	

SECTION 8

Termination

8

The termination routines cancel requests for supervisor operations and free the resources and control blocks associated with a terminating task. The requests and resources include:

- Enqueued resource requests
- Unexpired timer requests
- Incomplete operator communications
- Exclusively used programs in virtual storage
- Exclusively used data sets
- Unshared subpools of virtual storage

The control blocks that are removed from their queues and freed include one or more:

- Task control blocks (TCBs)
- Request blocks (RBs)
- Interruption queue elements (IQEs)
- Supervisor queue elements (SQEs)
- Queue elements (QELs)
- Subpool queue elements (SPQEs)
- Queue control blocks (QCBs)
- Contents directory entries (CDEs)
- Timer queue elements (TQEs)
- Terminal attention exit elements (TAXEs)
- The program interruption element (PIE) for the task, if one exists
- Data extent blocks (DEBs)

There are two types of termination procedures: normal and abnormal. Normal termination occurs when a task is complete; the last program to be executed for the task has completed processing. Abnormal termination occurs when some kind of unrecoverable error, such as an I/O error or program check, has taken place. The task must be terminated to prevent waste of system resources.

Normal and abnormal termination produce different results. Normal termination frees resources only for the completed task since it has no subtasks. Abnormal termination allows two options: task and step termination. In task termination, only the resources of the failing task and its subtasks are freed. In step termination, the resources used for the entire job step are freed, and, depending on the JCL, the job scheduler ignores later steps of the same job.

The general flow of the termination routines is shown in Diagram 8.1.

NORMAL TERMINATION -- EOT (END OF TASK)

(See Diagram 8.2)

When the last program to be executed for a task ends, it returns control to the Exit routine (SVC 3). The Exit routine branches to EOT (end-of-task) routine to free resources belonging to the terminating task, to schedule an ETXR (end-of-task exit routine) if one is specified, to post an end-of-task ECB (event control block) if one was specified when the task was created, and, if necessary, to ensure a task switch.

In its initial processing, EOT checks to see if the terminating task is in a "must complete" state, or if there are incomplete subtasks of the terminating task. If either abnormal condition exists, EOT forces abnormal termination by calling the ABTERM routine.

EOT dequeues enqueued resources, sets the completion code, and releases resources used by the terminating task. To accomplish these things, the routine does the following:

- Releases the PIE (program interruption element) if one exists.
- Purges unexpired TQEs (timer queue elements).
- Purges TAXEs (terminal attention exit elements).
- Purges operator communication queues.
- Closes data sets.
- Purges asynchronous exits (IQEs). if owned by the terminating task,

- Frees the task's last-executed program (or schedules it for a waiting requestor).
- Purges paging activity and migration.
- Releases loaded programs and the LLE (load list element) associated with each program.
- Purges fixed pages.
- Frees storage acquired for the task, and SPQEs (subpool queue elements) owned by the task.
- Dequeues the task's TCB (task control block) from the TCB priority queue.

If an ETXR was specified by the attaching task, EOT schedules it. If an ECB was specified by the attaching task, EOT schedules it for posting. When either an ETXR or ECB was specified, EOT also ensures that a task switch will be made and returns control to the Exit routine. If neither an ETXR nor an ECB was specified, EOT dequeues and frees the TCB and the last RB, frees the LSQA, if owned by the terminating task, ensures a task switch, and passes control to the dispatcher.

SCHEDULING ABNORMAL TERMINATION -- ABTERM

(See Diagram 8.10)

The ABTERM routine schedules abnormal termination for system routines that detect an error but cannot themselves issue an ABEND macro instruction. (See OS/VS Supervisor Services and Macro Instructions for a description of the ABEND macro instruction.) ABTERM also schedules abnormal termination for system routines that wish to terminate a task other than the one they are part of.

ABTERM performs the following major functions:

- Refreshes the CVT address.
- Saves the address of the next executable instruction and the general registers; they are displayed by ABDUMP during ABEND processing.
- Stores the completion code and dump option for ABEND.
- Builds a TIRB (task interruption request block) to cause remaining processing to be done under the TCB specified for termination.
- Schedules abnormal termination of the specified task by initializing an SQE with the address of an SVC 13 (ABEND) instruction in the CVT.

ABNORMAL TERMINATION -- ABEND

(See Diagram 8.13)

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.

The ABEND routine, which performs abnormal termination, 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 a system routine, after detecting an abnormal condition, branches to the ABTERM routine. The ABTERM routine schedules the execution of an SVC 13 instruction for the task to be terminated. The SVC 13 instruction, which is executed the next time the task to be terminated is dispatched, causes supervisor-assisted linkage to the ABEND routine.

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. This option permits a program belonging to a higher-level task in the job step to either continue or terminate the other tasks of the job step. 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 this task. Task termination of the job-step task, the highest-level task in the job step, produces the same result as a step termination.

If the failing task has issued a STAE macro instruction or was attached with the STAI option, the ABEND routine passes control to the ABEND/STA Interface routine (ASIR). The STAE/STAI exit routine specified by the requester is then given control by the requester is then given control ASIR. See OS/VS Supervisor Services and Macro Instructions for a description of the STAE macro instruction.

The following ciritical system tasks have STAR (System Task ABEND Recovery) routines which receive control from ASIR if the critical system task fails:

- Communications task
- Master scheduler task
- System error task
- IORMS (I/O Recovery Management Support) task

The STAR routines attempt recovery by specifying that the system task on whose behalf they are processing be retried. Both the ABEND and ASIR routines must recognize the failure of a critical system task and must pass control immediately to the appropriate STAR routine.

If the dump option has been selected (either by the user program or the ABTERM routine), the ABEND routine invokes the ABDUMP routine. The ABDUMP routine displays the programs, control blocks, and dynamically acquired storage of the terminating task, its descendants, and its immediate antecedents, including the job-step task.

An error condition may occur during ABEND processing that results in a new request for abnormal termination of the task that is already being terminated. The ABEND routine must be reentered to try to complete termination procedures. Reentering the ABEND routine for the same task is called a recursion. In some instances the ABEND routine expects the possibility of an ABEND failure, and these instances represent valid recursions; the ABEND routine processes the second request using any special handling necessary. Invalid recursions are ABEND failures that were not anticipated. They are considered to be ciritical errors; the reliability of the system is assumed to be in jeopardy when one occurs. A critical error causes the current task to be set nondispatchable; the job-step task is then scheduled for abnormal termination.

DUMPING SPECIFIED AREAS OF VIRTUAL STORAGE -- SVCDUMP

(See Diagram 8.25)

The SVCDUMP routine is invoked by an SVC 51 which is issued in a SNAP or SDUMP macro instruction. (See OS/VS Supervisor Services and Macro Instructions for a description of the SNAP macro instruction.) SVCDUMP can be called only by a program with protection key 0. If a caller's key is not 0, or if the caller requests, SVCDUMP passes control to ABDUMP.

SVCDUMP provides an unformatted dump of any area of virtual storage. If the caller specifies a TCB, the dump is for that task, rather than for the task requesting the dump. The dump is written to tape or disk, and can be printed using the system utility AMDPRDMP. (See OS/VS Service Aids for a description of AMDPRDMP.)

The caller may specify the areas of virtual storage he wishes dumped by providing SVCDUMP with a list of virtual storage addresses or by specifying any or all of the following areas (if the caller specifies no areas to be dumped, SVCDUMP displays all of the following):

- The nucleus.
- SQA (system queue area).
- The region and its LSQA (local system queue area).
- IPA (link pack area) modules associated with the task.

DUMPING SELECTED AREAS OF VIRTUAL STORAGE -- ABDUMP

(See Diagram 8.25)

The ABDUMP routine is invoked by a SNAP macro instruction. The SNAP macro instruction, whose expansion contains an SVC 51, causes the SVC Second-Level Interruption Handler (SVC SLIH) to call the SVCDUMP routine. The SVCDUMP routine checks the ABDUMP parameter list to determine whether a SNAP macro instruction has been issued. If so, the SVCDUMP routine passes control to the ABDUMP routine.

The SNAP macro instruction can be issued by the ABEND routine during abnormal termination, or by a user program at any time. Thus, ABDUMP processing can provide either a formatted abnormal dump or a formatted dynamic dump. The ABEND routine can specify either a SYSUDUMP or a SYSABEND dump. A SYSUDUMP dump consists of major control blocks belonging to the terminating task, its subtasks, and its immediate antecedents. Programs and dynamically acquired storage belonging to the terminating task are also displayed. A SYSABEND dump provides the same information as a SYSUDUMP dump with the following additions: the nucleus, LSQA (local system queue area), SQA (system queue area), load modules, subpool blocks, and the optional GTF (generalized trace facility) trace.

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/VS Supervisor Services and Macro Instructions_ for information on how to obtain a dump.)

The use of the ABDUMP routine is restricted to tasks that do not have job-step tasks within their subtask structure at entry to ABDUMP processing. If a task has a subtask that is a job-step task, control is returned immediately to the caller.

STA SERVICES AND ASIR (ABEND/STA INTERFACE ROUTINE)

(See Diagram 8.42)

The STA Services and ASIR routines allow a user's abnormal-end exit routine to analyze errors before abnormal termination proceeds, and if possible, to schedule a retry of a failing routine.

The STA Services routine is invoked by an SVC 60 issued in a STAE macro instruction or by the ATTACH service routine when the ATTACH macro has been issued with the "STAI= " operand. The routine creates, cancels, or overlays an SCB (STA control block) on a task's queue of SCBs, to serve as a means of communication between the STA Services, ABEND, and ASIR routines.

There are three kinds of SCBs:

- A _STAE SCB_ (there can be several on an SCB queue) specifies an exit routine to receive control when its associated task terminates abnormally.

- A _STAI SCB_ (there can be several on an SCB queue) specifies an exit routine (specified by the attaching task) to receive control when the attached task terminates abnormally.

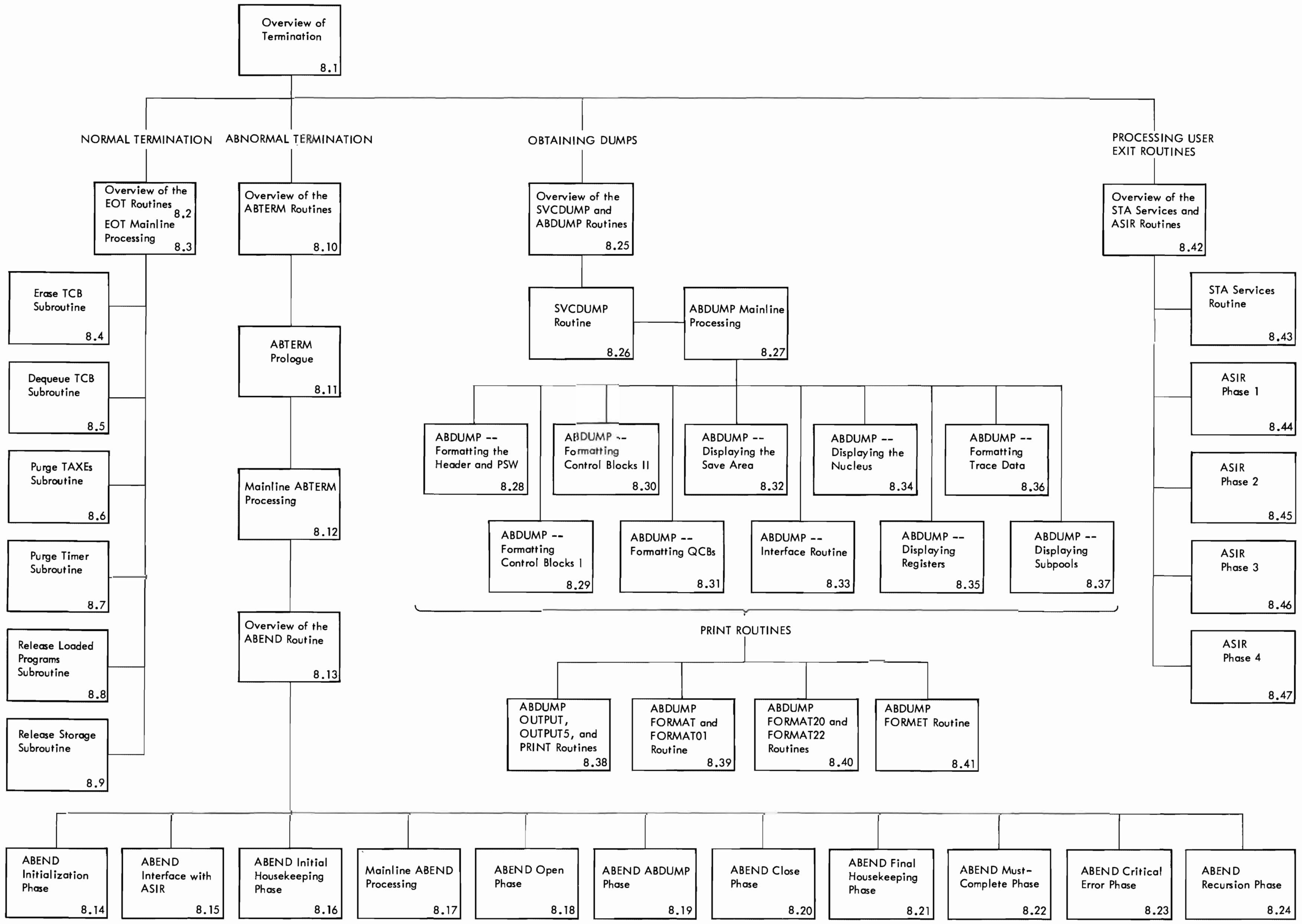
- A _STAR SCB_ (there can be only one STAR on the SCB queue) specifies an exit routine to receive control when a cirtical system task fails, but it receives control only after STAE and STAI exit routines have been tried. All permanent system tasks except the paging supervisor have STAR SCBs.

Index to Diagrams by Module Name for Termination

Module Name	Diagram Number(s)
IEAVAB00	8.10--8.12
IEAVAD00	8.26
IEAVAD01	8.27
IEAVAD02	8.28
IEAVAD03	8.29
IEAVAD05	8.30
IEAVAD06	8.31
IEAVAD07	8.32
IEAVAD08	8.33
IEAVAD0A	8.34
IEAVAD0B	8.35
IEAVAD0C	8.36
IEAVAD0D	8.37
IEAVAD11	8.38

Module Name	Diagram Number(s)
IEAVAD31	8.39
IEAVAD51	8.40
IEAVAD71	8.41
IEAVET00	8.2--8.9
IEAVSTA0	8.43
IEAVTM00	8.14--8.17, 8.24
IEAVTM01	8.18, 8.19
IEAVTM02	8.20, 8.21
IEAVTM03	8.22, 8.23
IEAVTM04	8.20
IEAVTM0B	8.44--8.46
IEAVTM0C	8.46, 8.47

Diagram 8.0
Termination
Visual Table of Contents



Overview of Termination 8.1
NORMAL TERMINATION
ABNORMAL TERMINATION
OBTAINING DUMPS
PROCESSING USER EXIT ROUTINES
Overview of the EOT Routines 8.2
EOT Mainline Processing 8.3
Erase TCB Subroutine 8.4
Dequeue TCB Subroutine 8.5
Purge TAXEs Subroutine 8.6
Purge Timer Subroutine 8.7
Release Loaded Programs Subroutine 8.8
Release Storage Subroutine 8.9
Overview of the ABTERM Routines 8.10
ABTERM Prologue 8.11
Mainline ABTERM Processing 8.12
Overview of the ABEND Routine 8.13
Overview of the SVCDUMP and ABDUMP Routines 8.25
SVCDUMP Routine 8.26
ABDUMP Mainline Processing 8.27
ABDUMP -- Formatting the Header and PSW 8.28
ABDUMP -- Formatting Control Blocks I 8.29
ABDUMP -- Formatting Control Blocks II 8.30
ABDUMP -- Formatting QCBs 8.31
ABDUMP -- Displaying the Save Area 8.32
ABDUMP -- Interface Routine 8.33
ABDUMP -- Displaying the Nucleus 8.34
ABDUMP -- Displaying Registers 8.35
ABDUMP -- Formatting Trace Data 8.36
ABDUMP -- Displaying Subpools 8.37
PRINT ROUTINES
ABDUMP OUTPUT, OUTPUT5, and PRINT Routines 8.38
ABDUMP FORMAT and FORMAT01 Routine 8.39
ABDUMP FORMAT20 and FORMAT22 Routines 8.40
ABDUMP FORMET Routine 8.41
Overview of the STA Services and ASIR Routines 8.42
STA Services Routine 8.43
ASIR Phase 1 8.44
ASIR Phase 2 8.45
ASIR Phase 3 8.46
ASIR Phase 4 8.47
ABEND Initialization Phase 8.14
ABEND Interface with ASIR 8.15
ABEND Initial Housekeeping Phase 8.16
Mainline ABEND Processing 8.17
ABEND Open Phase 8.18
ABEND ABDUMP Phase 8.19
ABEND Close Phase 8.20
ABEND Final Housekeeping Phase 8.21
ABEND Must-Complete Phase 8.22
ABEND Critical Error Phase 8.23
ABEND Recursion Phase 8.24

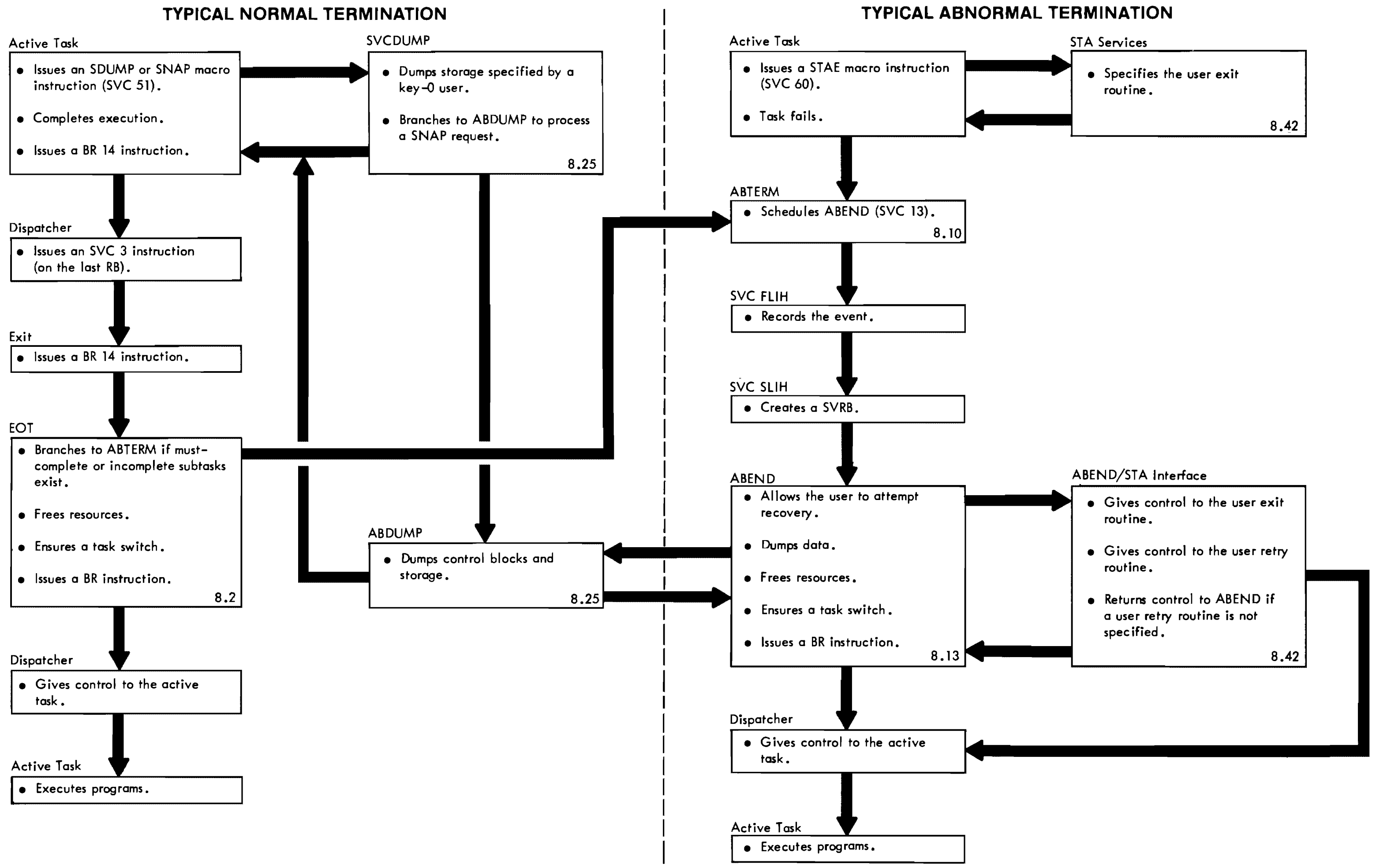
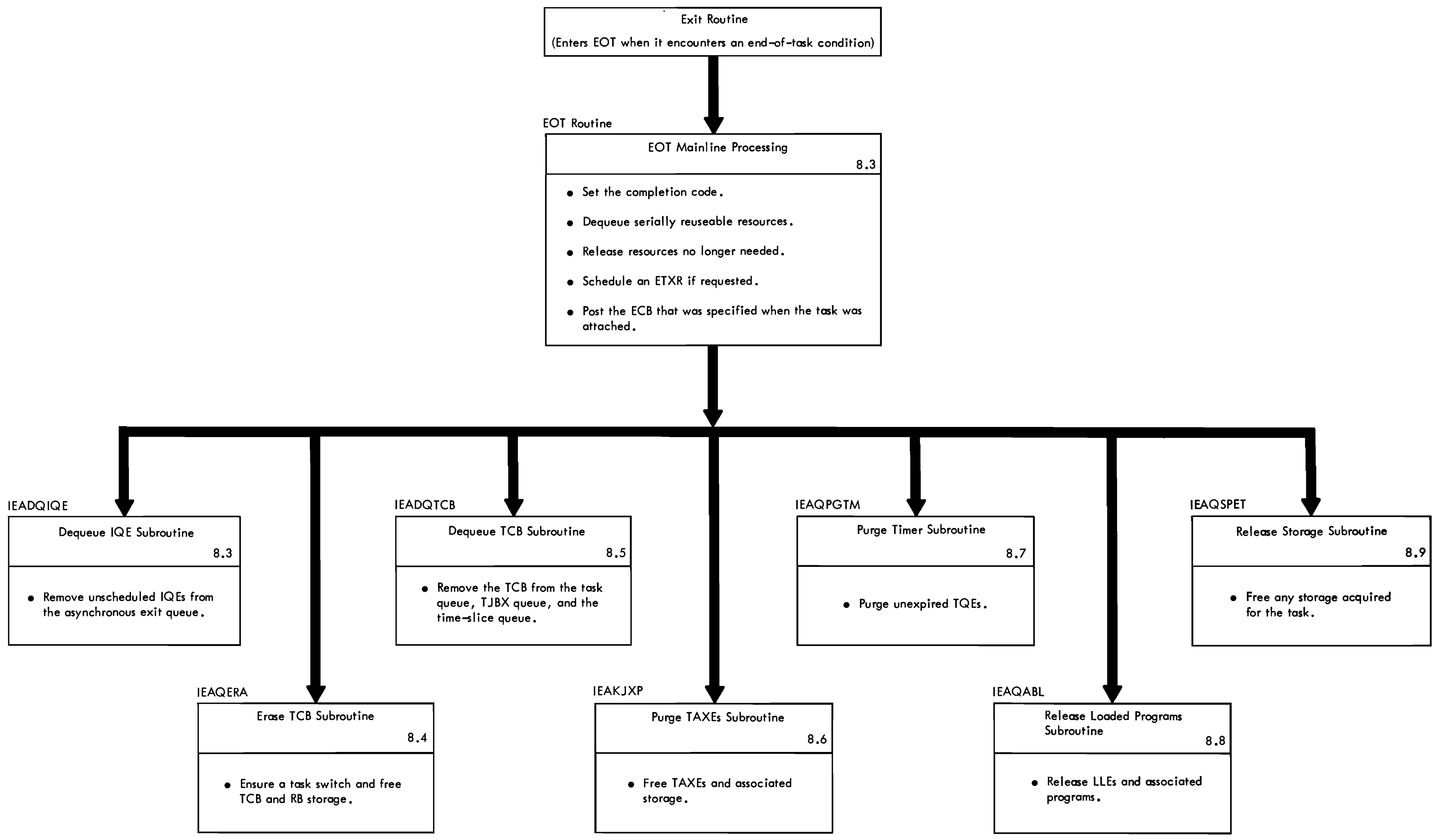
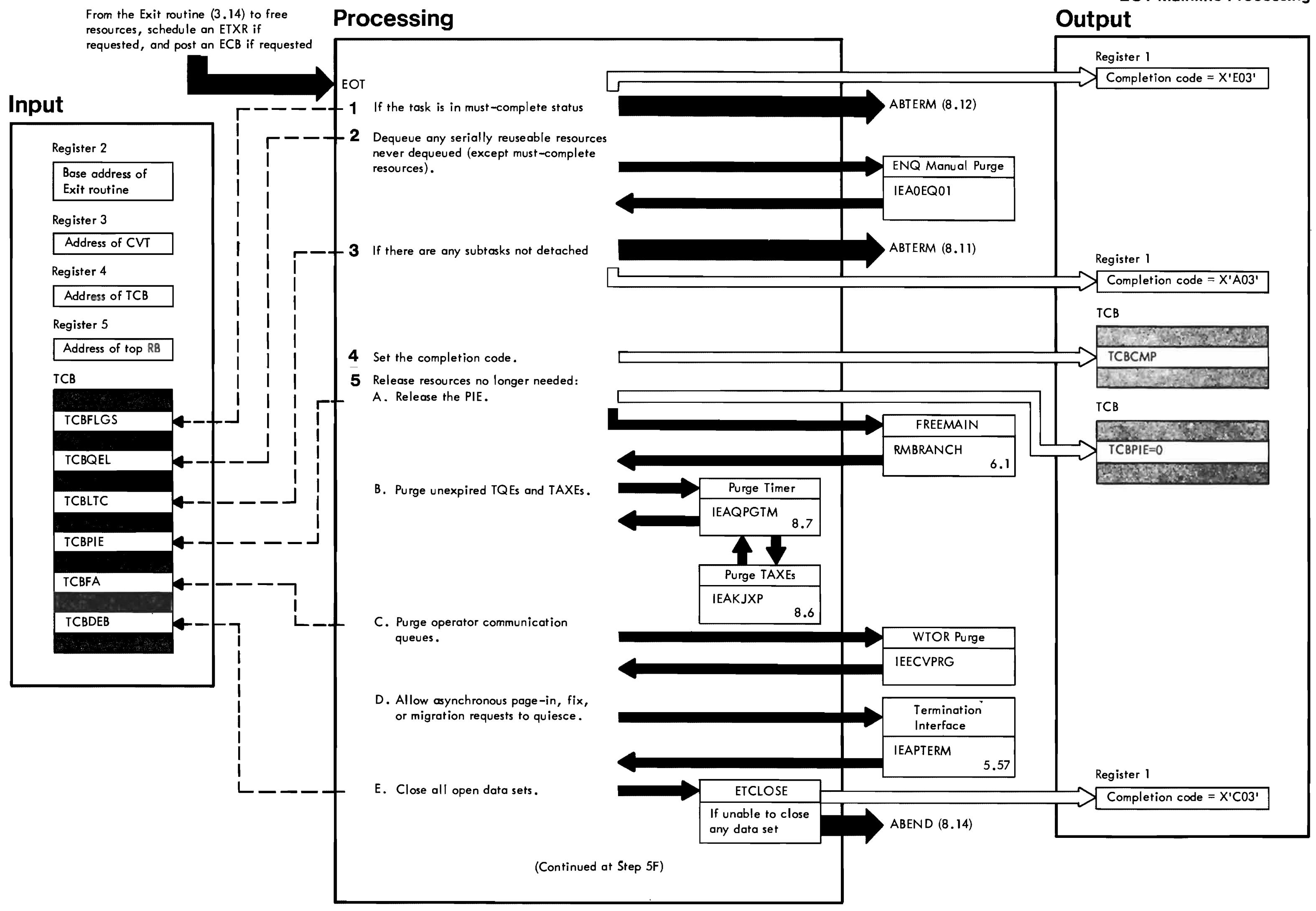

TYPICAL NORMAL TERMINATION
Active Task
• Issues an SDUMP or SNAP macro instruction (SVC 51).
• Completes execution.
• Issues a BR 14 instruction.
SVCDUMP
• Dumps storage specified by a key-0 user.
• Branches to ABDUMP to process a SNAP request.
8.25
Dispatcher
• Issues an SVC 3 instruction (on the last RB).
Exit
• Issues a BR 14 instruction.
EOT
• Branches to ABTERM if must-complete or incomplete subtasks exist.
• Frees resources.
• Ensures a task switch.
• Issues a BR instruction.
8.2
ABDUMP
• Dumps control blocks and storage.
8.25
Dispatcher
• Gives control to the active task.
Active Task
• Executes programs.
TYPICAL ABNORMAL TERMINATION
Active Task
• Issues a STAE macro instruction (SVC 60).
• Task fails.
STA Services
• Specifies the user exit routine.
8.42
ABTERM
• Schedules ABEND (SVC 13).
8.10
SVC FLIH
• Records the event.
SVC SLIH
• Creates a SVRB.
ABEND
• Allows the user to attempt recovery.
• Dumps data.
• Frees resources.
• Ensures a task switch.
• Issues a BR instruction.
8.13
ABEND/STA Interface
• Gives control to the user exit routine.
• Gives control to the user retry routine.
• Returns control to ABEND if a user retry routine is not specified.
8.42
Dispatcher
• Gives control to the active task.
Active Task
• Executes programs.

Diagram 8.2
Overview of EOT Routines



Exit Routine
(Enters EOT when it encounters an end-of-task condition)
EOT Routine
EOT Mainline Processing
8.3
• Set the completion code.
• Dequeue serially reuseable resources.
• Release resources no longer needed.
• Schedule an ETXR if requested.
• Post the ECB that was specified when the task was attached.
IEADQIQE
Dequeue IQE Subroutine
8.3
• Remove unscheduled IQEs from the asynchronous exit queue.
IEAQERA
Erase TCB Subroutine
8.4
• Ensure a task switch and free TCB and RB storage.
IEADQTCB
Dequeue TCB Subroutine
8.5
• Remove the TCB from the task queue, TJBX queue, and the time-slice queue.
IEAKJXP
Purge TAXEs Subroutine
8.6
• Free TAXEs and associated storage.
IEAQPGTM
Purge Timer Subroutine
8.7
• Purge unexpired TQEs.
IEAQABL
Release Loaded Programs Subroutine
8.8
• Release LLEs and associated programs.
IEAQSPET
Release Storage Subroutine
8.9
• Free any storage acquired for the task.

Diagram 8.3 (Steps 1-5E) EOT Mainline Processing



1 If the task is in must-complete status

2 Dequeue any serially reuseable resources never dequeued (except must-complete resources).

3 If there are any subtasks not detached

4 Set the completion code.

5 Release resources no longer needed:

A. Release the PIE.

B. Purge unexpired TQEs and TAXEs.

C. Purge operator communication queues.

D. Allow asynchronous page-in, fix, or migration requests to quiesce.

E. Close all open data sets.

(Continued at Step 5F)

Diagram 8.3 (Steps 1-5E) EOT Mainline Processing (Module IEAVETOO)

NOTES	ROUTINE NAME	LABEL
1 TCBFLGS, set with TCBFJMC, or TCBFSMC means that this task is in must-complete status.	EOT	
2 TCBQEL, when set, means the terminating task is enqueued on resources. (See "Section 3: Task Supervision" for information on the ENQ Manual Purge routine.)		
3 TCBLTC is the subtask pointer. It should contain 0.		ERENQ
4 The task that attached the terminating task checks the completion code to determine the subtask's status, but only if the terminating task was attached with the ECB or ETXR operand.		
5 A. ETCLOSE issues a GETMAIN request for a CLOSE parameter list and issues a CLOSE macro instruction for each data set on the TCBDEB queue. TCBPIE contains the address of the PIE, or 0 if there is no PIE. It is set by the SPIE routine.		
B. A TQE represents a request for a timer interval that has not expired.		
C. TCBFA, when set, means the task has abnormally terminated, in which case the WTOR purge is bypassed now and performed later by ABEND. The WTOR Purge routine requests the operator to cancel outstanding replies to messages from this task.		
For further information about the WTOR Purge routine, see <u>OS/VS Job Management Logic</u>.		
E. TCBDEB contains the address of the first DEB (data extent block). The DEB queue contains pointers to all DCBs for a task.		

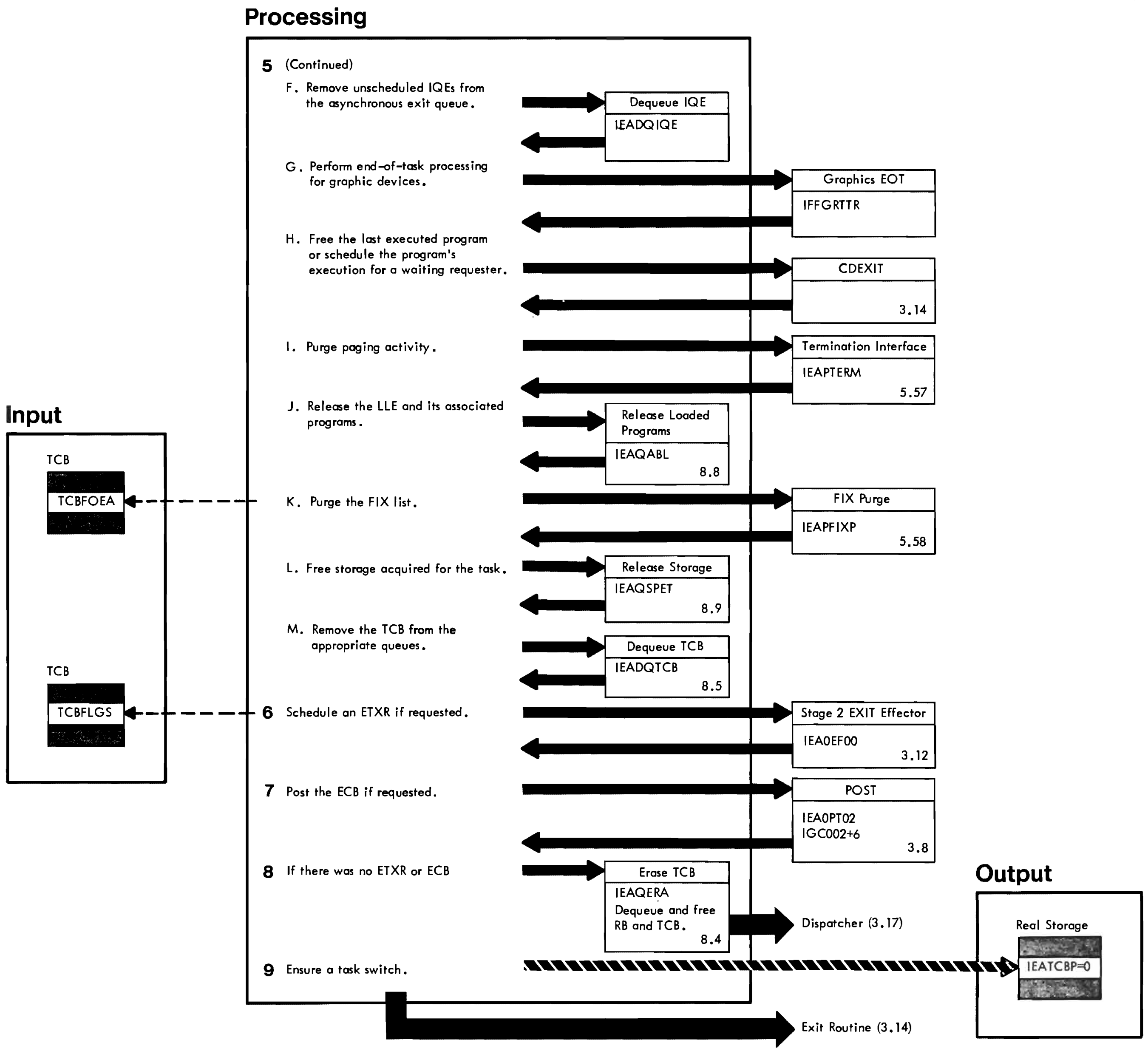
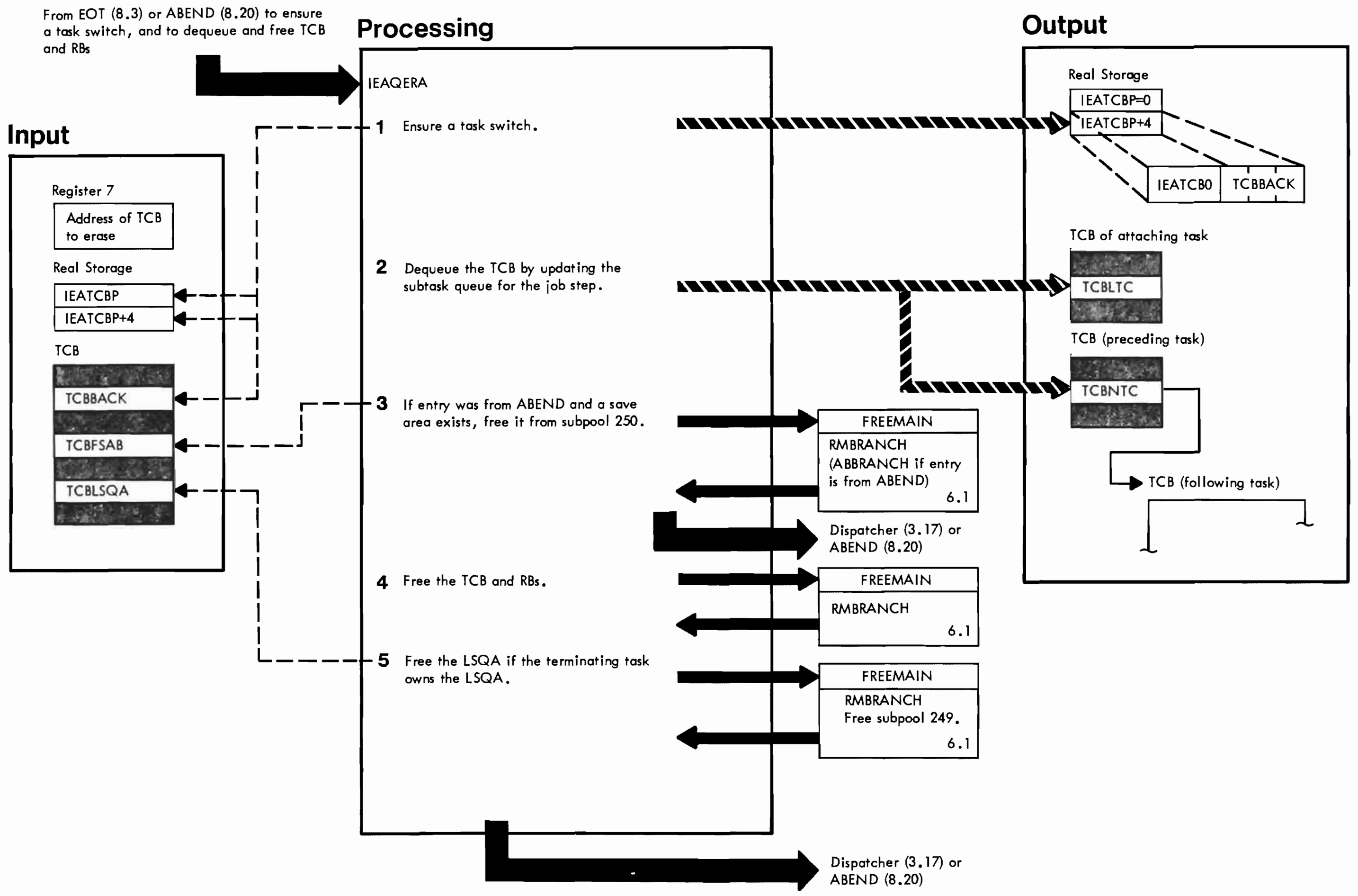

Input
TCB
TCBFOEA
TCB
TCBFLGS
Processing
5 (Continued)
F. Remove unscheduled IQEs from the asynchronous exit queue.
G. Perform end-of-task processing for graphic devices.
H. Free the last executed program or schedule the program's execution for a waiting requester.
I. Purge paging activity.
J. Release the LLE and its associated programs.
K. Purge the FIX list.
L. Free storage acquired for the task.
M. Remove the TCB from the appropriate queues.
6 Schedule an ETXR if requested.
7 Post the ECB if requested.
8 If there was no ETXR or ECB
9 Ensure a task switch.
Dequeue IQE
IEADQIQE
Graphics EOT
IFFGRTTR
CDEXIT
3.14
Termination Interface
IEAPTERM
5.57
Release Loaded Programs
IEAQABL
8.8
FIX Purge
IEAPFIXP
5.58
Release Storage
IEAQSPET
8.9
Dequeue TCB
IEADQTCB
8.5
Stage 2 EXIT Effector
IEAOEF00
3.12
POST
IEAOPT02
IGC002+6
3.8
Erase TCB
IEAQERA
Dequeue and free RB and TCB.
8.4
Dispatcher (3.17)
Exit Routine (3.14)
Output
Real Storage
IEATCBP=0

Diagram 8.3 (Steps 5F-9) EOT Mainline Processing (Module IEAVET00)

NOTES	ROUTINE NAME	LABEL
5 F. The Dequeue IQE subroutine removes any IQEs (interruption queue elements) belonging to the terminating task that have not yet been scheduled. After registers have been saved the asynchronous exit queue is located (FLCAEQJ) and the first IQE is obtained. If there are no IQEs, execution continues at label DONE1. If the IQE is for the terminating task, it is dequeued; otherwise, the next IQE is obtained. After each IQE has been dequeued, a test is made to determine whether the last IQE has been dequeued (FLCAEQK). If not, the next IQE is obtained; if yes, a check is made to ensure that the pointer to the last IQE is correct. Registers are restored and control returns to EOT Mainline processing.	IEADQIQE IEADQIQE IEADQIQE IEADQIQE	ETDQ ETDQ ETDQ DONE1
G. If graphic devices are not included in the system, the Graphics EOT routine immediately returns to EOT.	EOT	
For further information about the Graphics EOT routine, see <u>OS/VS Graphics Access Method Logic</u>.		
L. On return from the Release Storage subroutine, register 3 is reset to the address of the CVT in case the address was destroyed.		
6 TCBFLGS, when set with TCBFETXR, means an ETXR is requested.		
7 POST can be entered in two places:		
A. If the ECB is for a job-step task, entry point IGC002+6 is used.		
B. If the ECB is not for a job-step task, entry point IEA0PT02 is used to cause validity checking.		
9 Zeroing the new TCB pointer (IEATCBP) indicates to the dispatcher that it must search down the task queue to find the next higher-priority ready task. (The Exit routine frees the RB and branches to the dispatcher.)		

Diagram 8.4 Erase TCB Subroutine



From EOT (8.3) or ABEND (8.20) to ensure a task switch, and to dequeue and free TCB and RBs

1 Ensure a task switch.

2 Dequeue the TCB by updating the subtask queue for the job step.

3 If entry was from ABEND and a save area exists, free it from subpool 250.

4 Free the TCB and RBs.

5 Free the LSQA if the terminating task owns the LSQA.

Diagram 8.4 Erase TCB Subroutine (Module IEAVETOO)

NOTES	ROUTINE NAME	LABEL
1 IEATCBP+4 contains the address of the current (old) TCB. TCBBACK contains the address of the next higher-priority TCB. If the new (IEATCBP) and old (IEATCBP+4) pointers are equal, no higher-priority task is ready. The new TCB pointer is set to 0 as indicated and the old TCB pointer is set equal to TCBBACK. If the new and old pointers are not equal, a task switch is already indicated; only the old TCB pointer is set equal to TCBBACK. IEATCB0 is set in the flag byte of the old TCB pointer to prevent the dispatcher from saving the floating-point registers.	IEAQERA	
2 TCBLTC points to the last attached subtask. If the terminating task is the last attached subtask, TCBLTC in the attaching task must be set to equal the terminating task's preceding same-level subtask (TCBNTC in the terminating task's TCB). If the terminating task is not the last attached subtask, only the subtask chain must be changed: the TCBNTC field of the following subtask must be changed to equal the TCBNTC field of the terminating task.		
4 If the terminating task does not own the LSQA, the TCB is freed from the attaching task's subpool 253. If the terminating task does own the LSQA, the TCB is freed explicitly since it is embedded in the LSQA. It is freed in Step 6 when the LSQA is freed.		
5 TCBLSQA, when set, signifies that this task owns the LSQA, which should be freed. If entry is from ABEND, return is made to ABEND. Otherwise, the dispatcher receives control.		

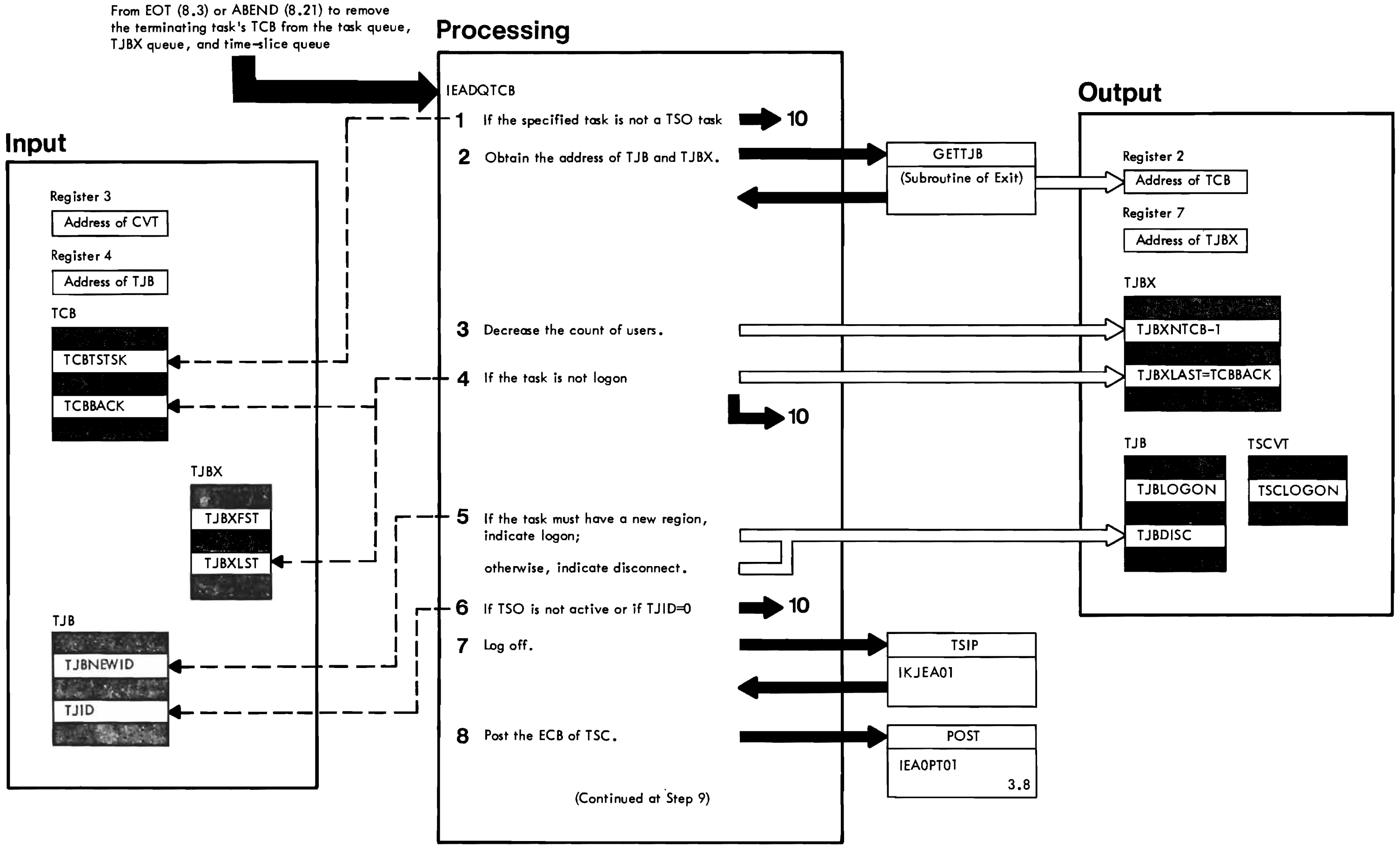
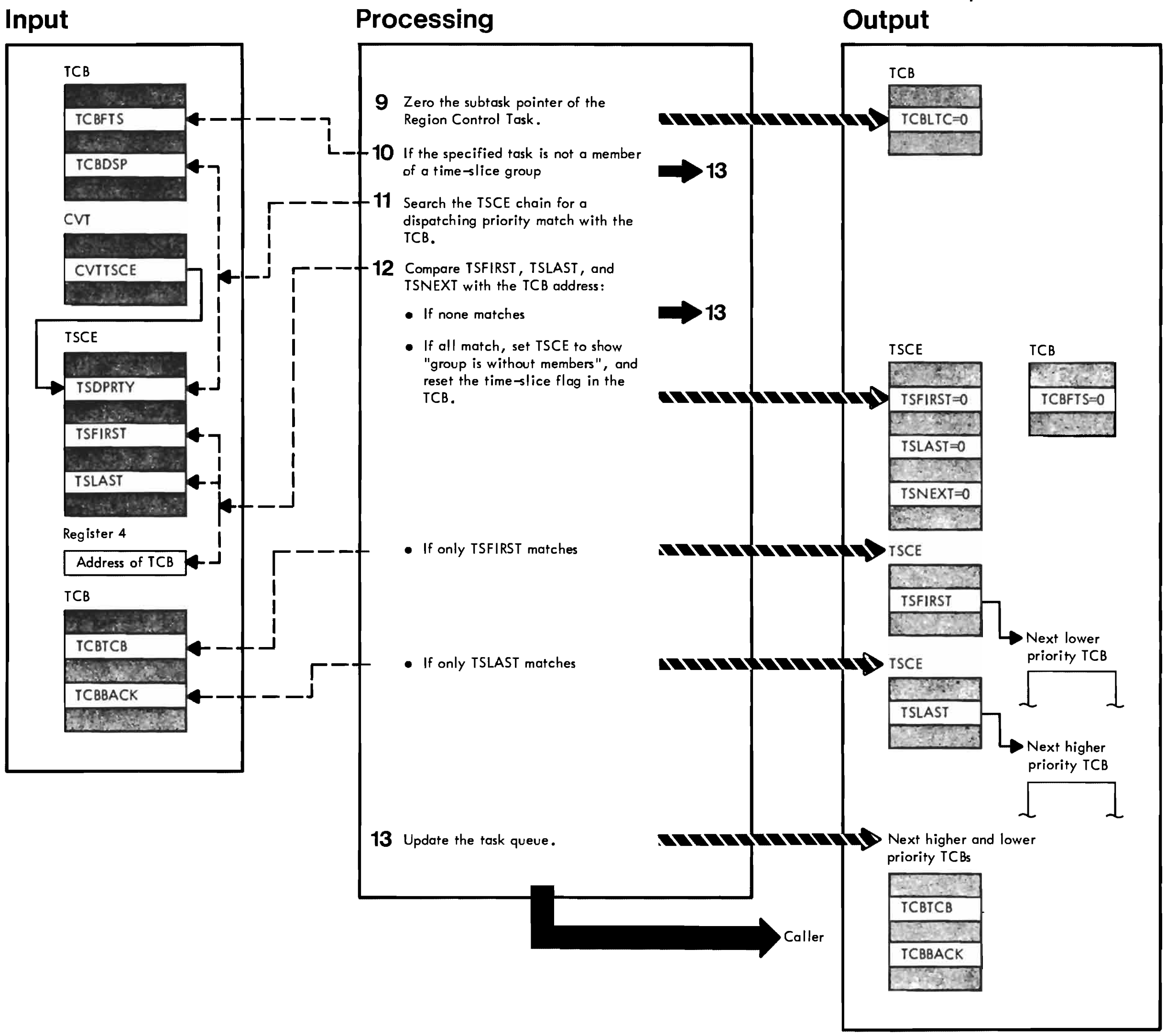

From EOT (8.3) or ABEND (8.21) to remove the terminating task's TCB from the task queue, TJBX queue, and time-slice queue
Input
Register 3
Address of CVT
Register 4
Address of TJB
TCB
TCBTSTSK
TCBBACK
TJBX
TJBXFST
TJBXLST
TJB
TJBNEWID
TJID
Processing
IEADQTCB
1 If the specified task is not a TSO task
10
2 Obtain the address of TJB and TJBX.
GETTJB
(Subroutine of Exit)
3 Decrease the count of users.
4 If the task is not logon
10
5 If the task must have a new region, indicate logon;
otherwise, indicate disconnect.
6 If TSO is not active or if TJID=0
10
7 Log off.
TSIP
IKJEA01
8 Post the ECB of TSC.
POST
IEAOPT01
3.8
(Continued at Step 9)
Output
Register 2
Address of TCB
Register 7
Address of TJBX
TJBX
TJBXNTCB-1
TJBXLAST=TCBBACK
TJB
TJBLOGON
TJBDISC
TSCVT
TSCLOGON

Diagram 8.5 (Steps 1-8) Dequeue TCB Subroutine (Module IEAVET00)

NOTES	ROUTINE NAME	LABEL
1 TCBTSTSK, when set, means that this is a TSO TCB.	IEADQTCB	TSOEOT
4 TJBXFST contains the address of the logon TCB. This field is compared with TJBXLST (the last TSO TCB); if the two are equal, only the logon TCB remains -- this is the logon task. TJBXLST needs to be updated only if it points to the terminating task's TCB.		
5 TJBNEWID, when set, means "assign a new region to this task."		LOGONXIT
6		CNTRELOG
7 For further information about the TSIP routine, see <u>OS/VS2 TSO Control Program Logic</u>.		
8 The address of the POST routine is obtained from the CVT (CVT0PT01).		



9 Zero the subtask pointer of the Region Control Task.

10 If the specified task is not a member of a time-slice group

11 Search the TSCE chain for a dispatching priority match with the TCB.

12 Compare TSFIRST, TSLAST, and TSNEXT with the TCB address:

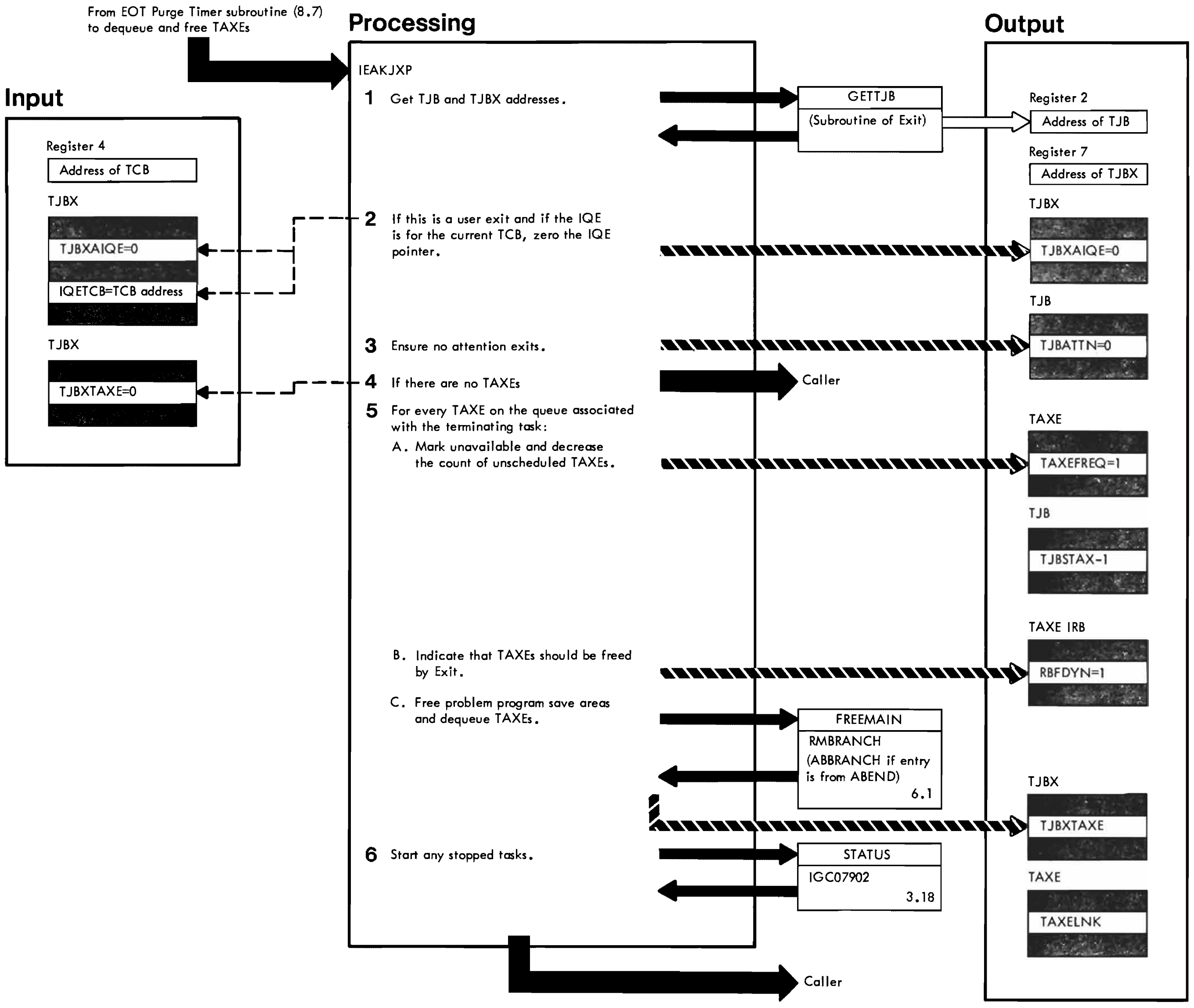
- If none matches
- If all match, set TSCE to show "group is without members", and reset the time-slice flag in the TCB.
- If only TSFIRST matches
- If only TSLAST matches

13 Update the task queue.

Diagram 8.5 (Steps 9-13) Dequeue TCB Subroutine (Module IEAVFT00)

NOTES	ROUTINE NAME	LABEL
9	IEADQTCB	RESTORE
10 TCBFTS, when set, means that this task is time-sliced.		TMSLTSK
11		FINDTSCE
12 TSFIRST and TSLAST point to the first and last TCBs of a time-slice group. TCBTCB is the address of the next lower-priority TCB. TCBBACK is the address of the next higher-priority TCB.		CHKFST
13 The TCBTCB field of the next higher-priority TCB is set to the TCBTCB field of the terminating TCB. The TCBBACK field of the next lower-priority TCB is set to the TCBBACK field of the terminating TCB. If the terminating task is the lowest APG (automatic priority group) task, IEATCBP+4 is set equal to TCBBACK and the high-order byte of IEATCBP+4 is set to X'80'.		ENDTMSL

Diagram 8.6
Purge TAXEs Subroutine



1 Get TJB and TJBX addresses.

2 If this is a user exit and if the IQE is for the current TCB, zero the IQE pointer.

3 Ensure no attention exits.

4 If there are no TAXEs

5 For every TAXE on the queue associated with the terminating task:

A. Mark unavailable and decrease the count of unscheduled TAXEs.

B. Indicate that TAXEs should be freed by Exit.

C. Free problem program save areas and dequeue TAXEs.

6 Start any stopped tasks.

Diagram 8.6 Purge TAXEs Subroutine (Module IEAVET00)

NOTES	ROUTINE NAME	LABEL
1	IEAKJXP	TAXEPRG
2 TJBXAIQE points to the IQE. It contains a 0 if this is a user exit.		
3 TJBATTN contains the TJB attention count.		NOSCHED
5 A. The TAXETCB in each associated TAXE contains the address of the current TCB.		TAXEFND
B.		SKPDECR
C.		NOACTV
6 There are no subtasks to start if EOT has called this routine. If ABEND is the caller, however, there may be subtasks that still must be dispatched.		

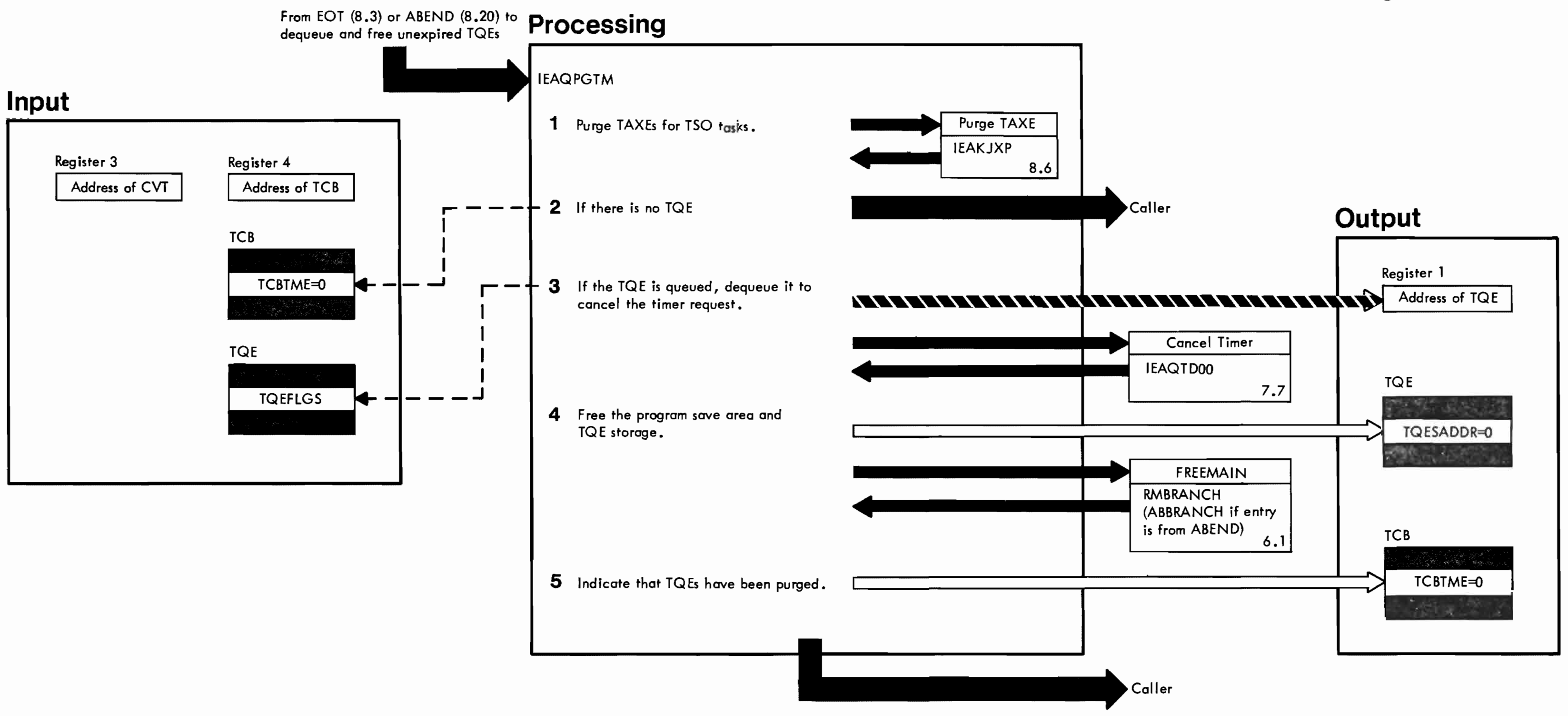
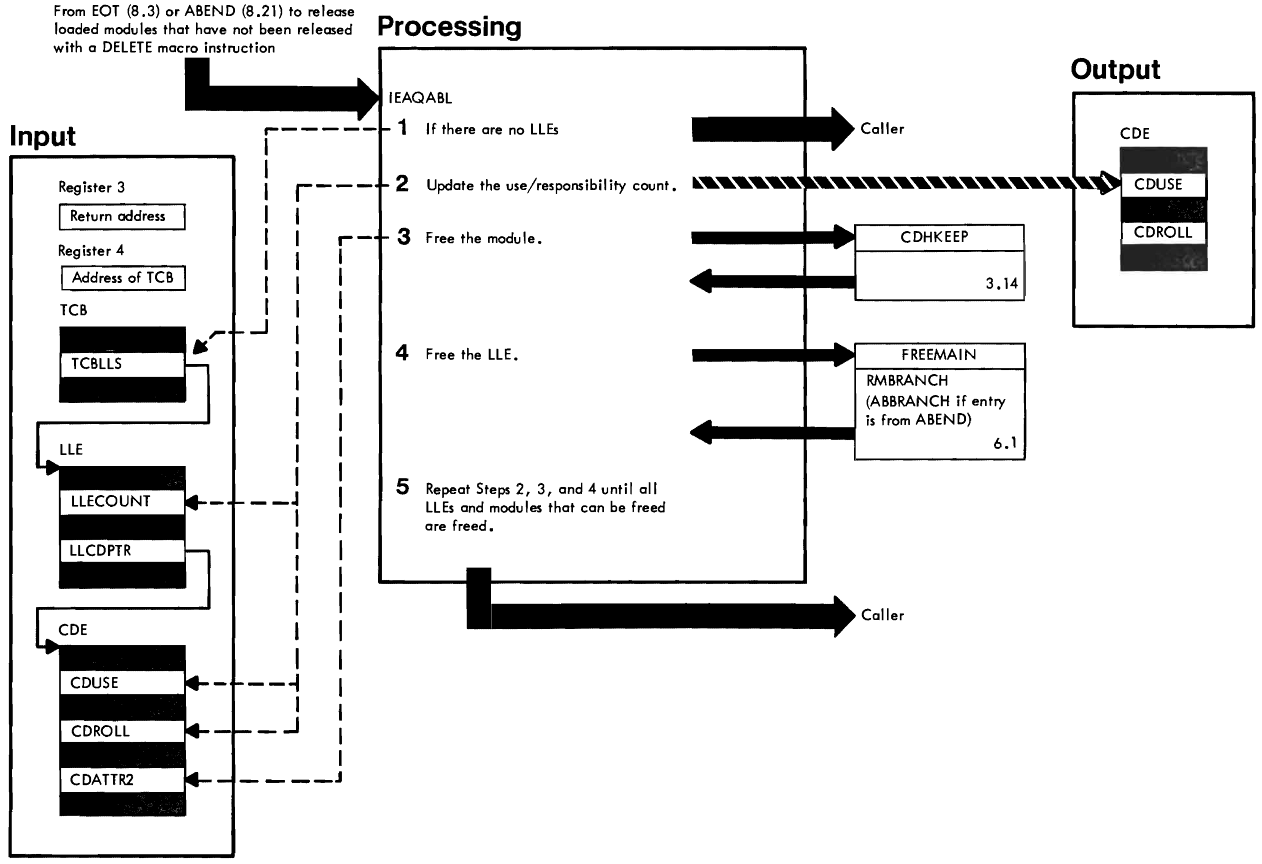

From EOT (8.3) or ABEND (8.20) to dequeue and free unexpired TQEs
Input
Register 3
Address of CVT
Register 4
Address of TCB
TCB
TCBTME=0
TQE
TQEFLGS
Processing
IEAQPGTM
1 Purge TAXEs for TSO tasks.
Purge TAXE
IEAKJXP
8.6
2 If there is no TQE
Caller
3 If the TQE is queued, dequeue it to cancel the timer request.
Cancel Timer
IEAQTD00
7.7
4 Free the program save area and TQE storage.
FREEMAIN
RMBRANCH (ABBRANCH if entry is from ABEND)
6.1
5 Indicate that TQEs have been purged.
Caller
Output
Register 1
Address of TQE
TQE
TQESADDR=0
TCB
TCBTME=0

Diagram 8.7 Purge Timer Subroutine (Module IEAVET00)

NOTES	ROUTINE NAME	LABEL
1	IEAQPGTM	ETMBSE
2 TCBTME=0 means that there is no TQE.		SKPTAXPG
3 If bit 0 of TQEFLGS is set, the TQE is not on the timer queue. The address of IEAQTD00 is obtained from the CVT (CVTQTD00).		
4 TQESADDR is zeroed to indicate that the save area is freed. The program save area is in subpool 250. TQEs are in subpool 253.		ETMFREE
5		ETQEFR

Diagram 8.8
Release Loaded Programs Subroutine



From EOT (8.3) or ABEND (8.21) to release loaded modules that have not been released with a DELETE macro instruction

IEAQABL

1 If there are no LLEs

2 Update the use/responsibility count.

3 Free the module.

4 Free the LLE.

5 Repeat Steps 2, 3, and 4 until all LLEs and modules that can be freed are freed.

Diagram 8.8 Release Loaded Programs Subroutine (Module IEAVET00)

NOTES	ROUTINE NAME	LABEL
1 If TCBLLS=0, there are no LLEs.	IEAQABL	ETLDP
2 LLECOUNT=number of load requests for a module.		ETCTADJ
CDUSE and CDROLL, viewed as a single field, contain the total number of requests for a module.		
NIP sets the use count of any fixed link pack area module to 1 so that when EOT decreases the use count, it never reaches 0.		
Each LLE points to an associated CDE.		
3 CDHKEEP is a subroutine of the Exit routine.		
Note: Modules on the job pack area queue are released by the Release Storage subroutine, Diagram 8.9.		

Diagram 8.9
Release Storage Subroutine

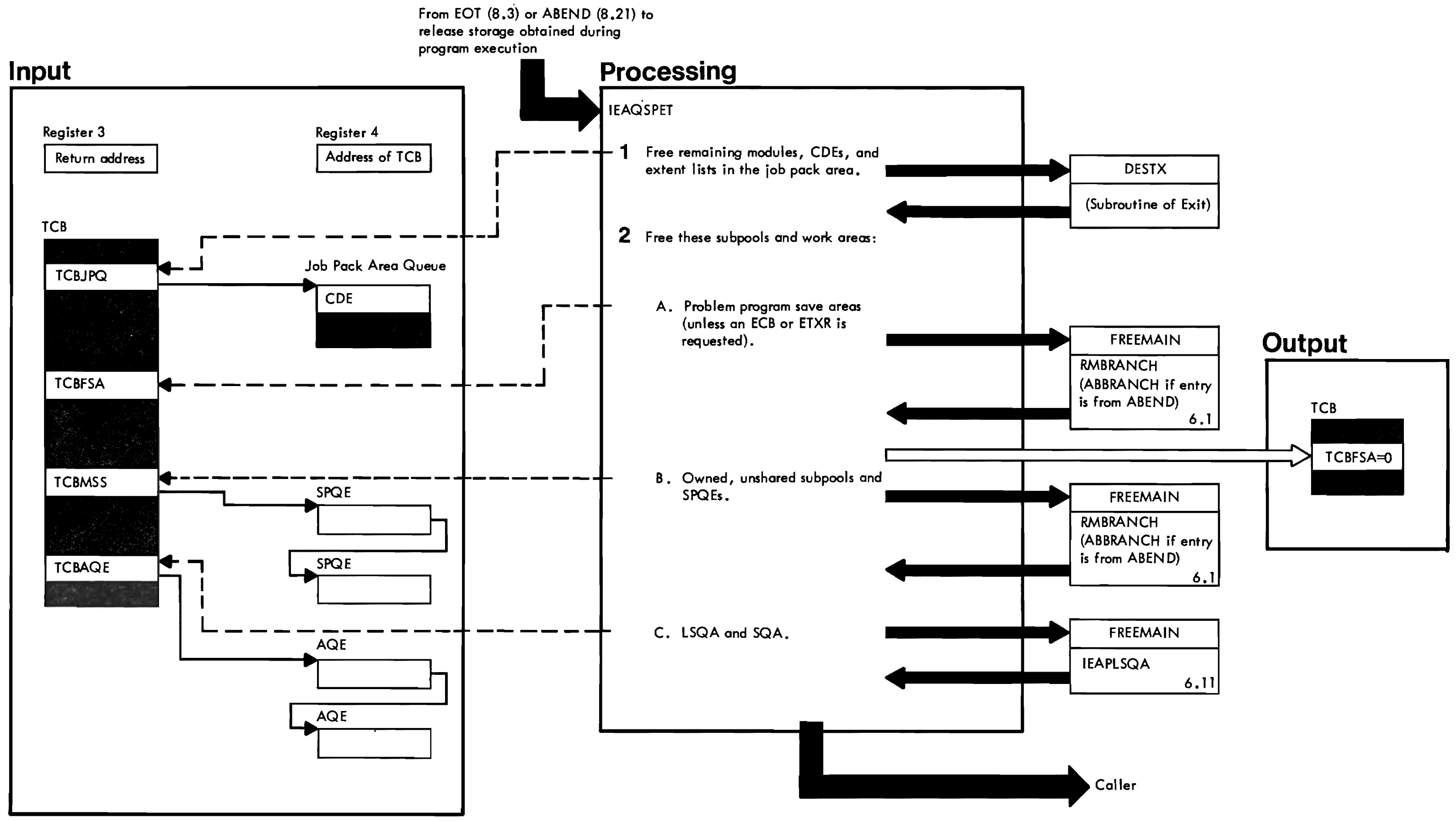
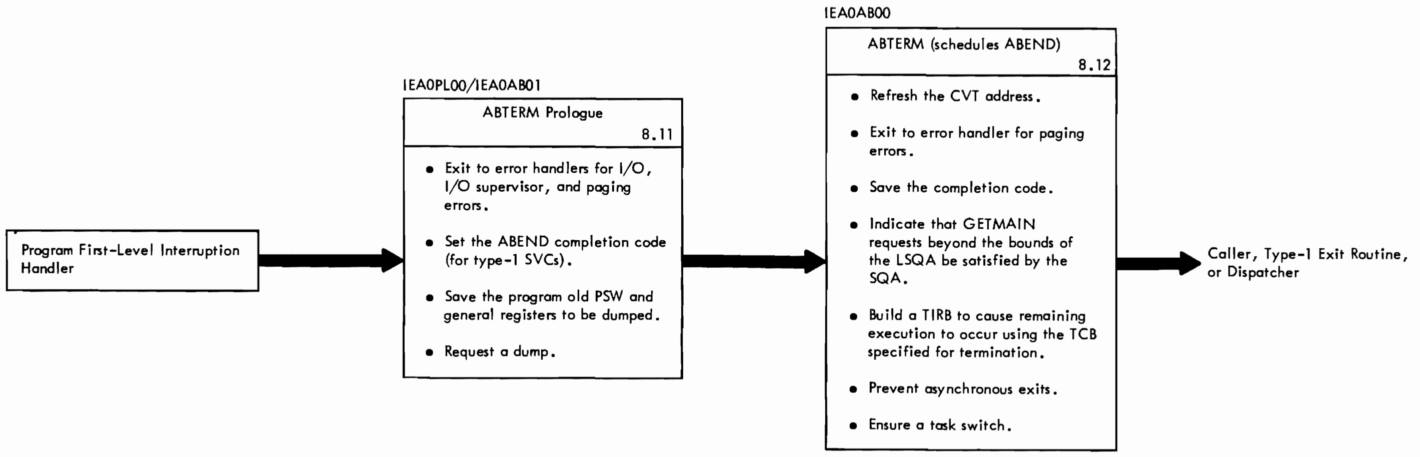

From EOT (8.3) or ABEND (8.21) to release storage obtained during program execution
Input
Register 3
Return address
Register 4
Address of TCB
TCB
TCBJPQ
Job Pack Area Queue
CDE
TCBFSA
TCBMSS
SPQE
SPQE
TCBAQE
AQE
AQE
Processing
IEAQSPET
1 Free remaining modules, CDEs, and extent lists in the job pack area.
2 Free these subpools and work areas:
A. Problem program save areas (unless an ECB or ETXR is requested).
B. Owned, unshared subpools and SPQEs.
C. LSQA and SQA.
DESTX
(Subroutine of Exit)
FREEMAIN
RMBRANCH (ABBRANCH if entry is from ABEND)
6.1
FREEMAIN
RMBRANCH (ABBRANCH if entry is from ABEND)
6.1
FREEMAIN
IEAPLSQA
6.11
Caller
Output
TCB
TCBFSA=0

Diagram 8.9 Release Storage Subroutine (Module IEAVET00)

NOTES	ROUTINE NAME	LABEL
1 The job pack area queue must contain at least one CDE. If it does not, Step 1 is bypassed.	IEAQSPET	ETMSLP
2 A. TCBFSA=0 means that there is no problem program save area. If an ECB or ETXR was specified, the save area is not freed. It must be retained for examination by the attaching task and is freed later by the DETACH routine.		ETFRST
B. Only Subpool storage that is owned by the terminating task is freed. SPQEs are dequeued before the storage occupied by the SPQE is freed.		ETFSPQE
C. Subpool 253 contains the TIRB, the parameter list for CLOSE, and any TAXEs and TQEs of the terminating task.		ETFAQE
If TCBAQE=0, there are no AQEs (allocated queue elements); this step is bypassed.		
Subpools 254 and 244 are automatically freed by FREEMAIN if the terminating task is the job-step task.		

Diagram 8.10
Overview of the ABTERM Routines



Program First-Level Interruption Handler

IEAOPL00/IEAOAB01

ABTERM Prologue 8.11

- Exit to error handlers for I/O, I/O supervisor, and paging errors.
- Set the ABEND completion code (for type-1 SVCs).
- Save the program old PSW and general registers to be dumped.
- Request a dump.

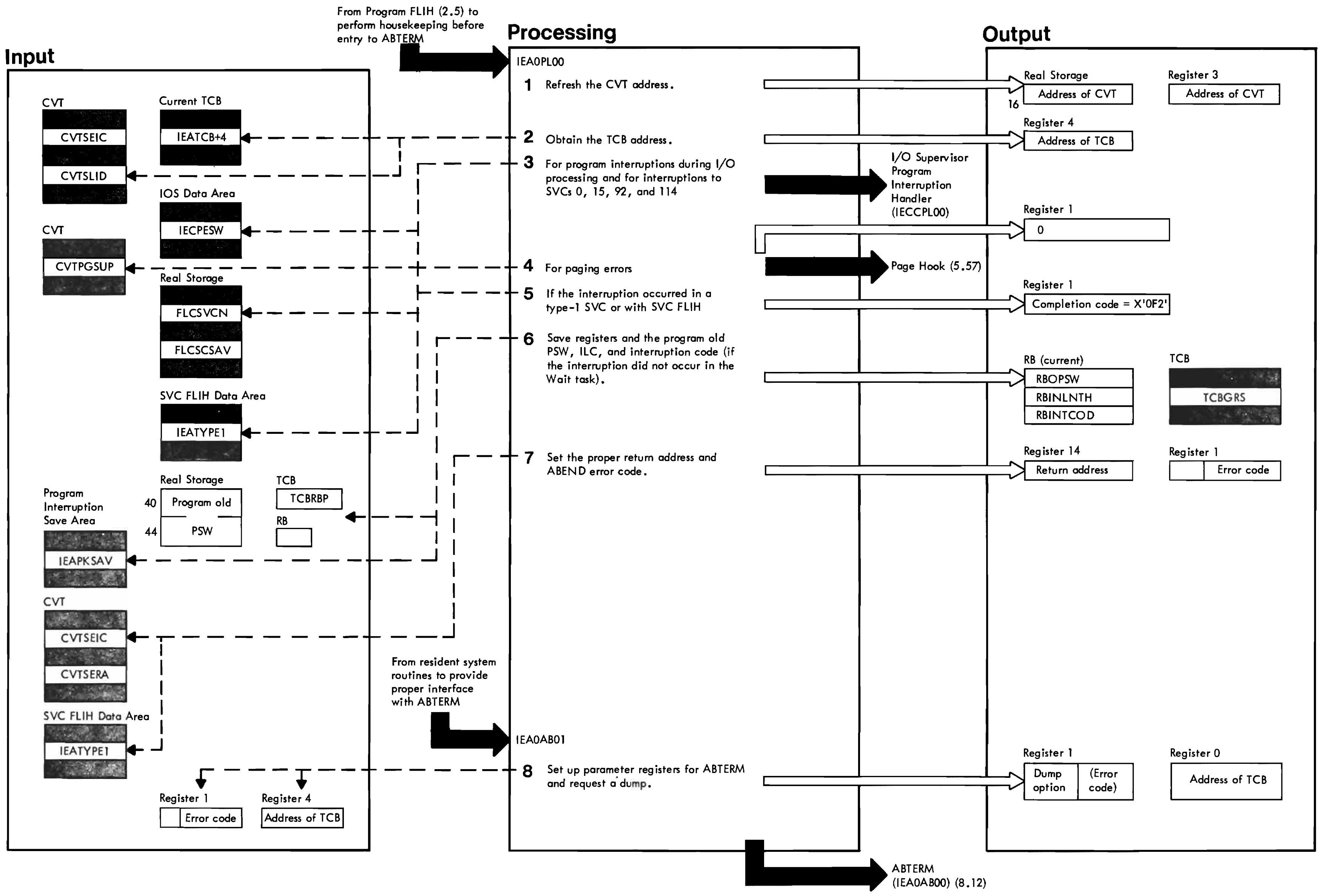
IEAOAB00

ABTERM (schedules ABEND) 8.12

- Refresh the CVT address.
- Exit to error handler for paging errors.
- Save the completion code.
- Indicate that GETMAIN requests beyond the bounds of the LSQA be satisfied by the SQA.
- Build a TIRB to cause remaining execution to occur using the TCB specified for termination.
- Prevent asynchronous exits.
- Ensure a task switch.

Caller, Type-1 Exit Routine, or Dispatcher

Diagram 8.11 ABTERM Prologue



1 Refresh the CVT address.

2 Obtain the TCB address.

3 For program interruptions during I/O processing and for interruptions to SVCs 0, 15, 92, and 114

4 For paging errors

5 If the interruption occurred in a type-1 SVC or with SVC FLIH

6 Save registers and the program old PSW, ILC, and interruption code (if the interruption did not occur in the Wait task).

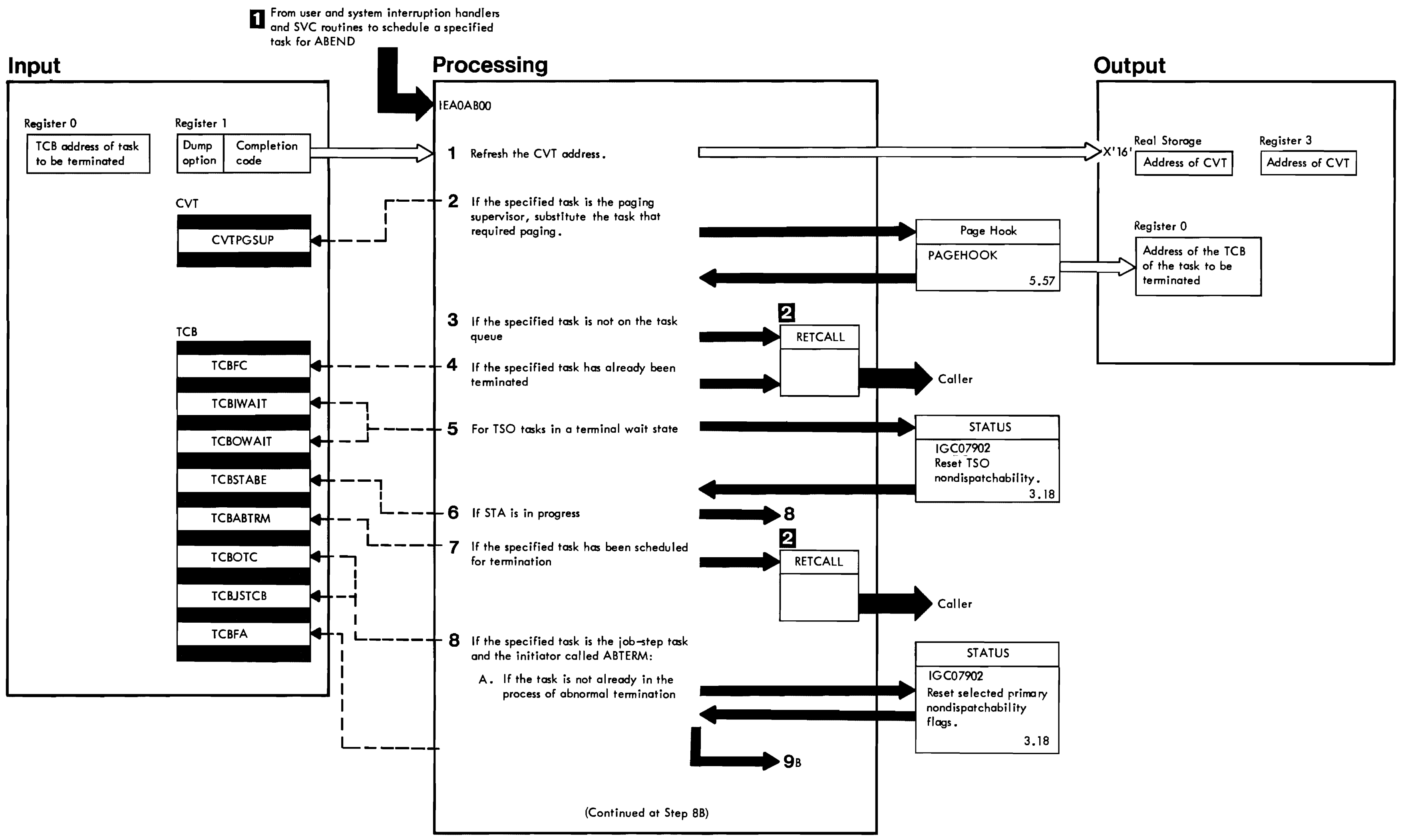
7 Set the proper return address and ABEND error code.

8 Set up parameter registers for ABTERM and request a dump.

Diagram 8.11 ABTERM Prologue (Module IEAVAB00)

NOTES	ROUTINE NAME	LABEL
1 The CVT address at location 16 is refreshed because lower real storage is especially vulnerable to being overlaid.	IEA0PL00	PROLOGUE
2 CVTSEIC is the second exit indicator. If set, the TCB address is obtained from CVTSLID. Otherwise, the current TCB address (IEATCBP+4) is obtained.		
3 IECPESW, when set, means that an interruption occurred during I/O processing. IEATYPE1 is checked to determine whether the interruption occurred in a type-1 SVC. FLCSVCN contains the SVC number. FLCSCSAV is the register save area. For further information about the IOS Program Interruption Handler, see <u>OS/VS I/O Supervisor Logic</u>.		IOSEXIT
4		PAGEXIT
5		PCTYPE1
6 Registers from the program check save area are moved into the TCB. If the interruption did not occur in the Wait task, the program check old PSW, the ILC (interruption length code), and the interruption code are saved in the current RB. This information appears later in the ABEND dump. If the interruption occurred in the Wait task, the old PSW cannot be saved because it would overlay the Wait PSW. The ILC and interruption code, if saved, would overlay the floating-point register save area.		ABSTAT1
7 If CVTSEIC is set, the return address is set equal to CVTSERA. If the type-1 switch (IEATYPE1) is set, the return address is set equal to the Type-1 Exit routine. Otherwise, the return address is set with the address of the dispatcher.		

Diagram 8.12 (Steps 1-8A) Mainline ABTERM Processing



1 Refresh the CVT address.

2 If the specified task is the paging supervisor, substitute the task that required paging.

3 If the specified task is not on the task queue

4 If the specified task has already been terminated

5 For TSO tasks in a terminal wait state

6 If STA is in progress

7 If the specified task has been scheduled for termination

8 If the specified task is the job-step task and the initiator called ABTERM:

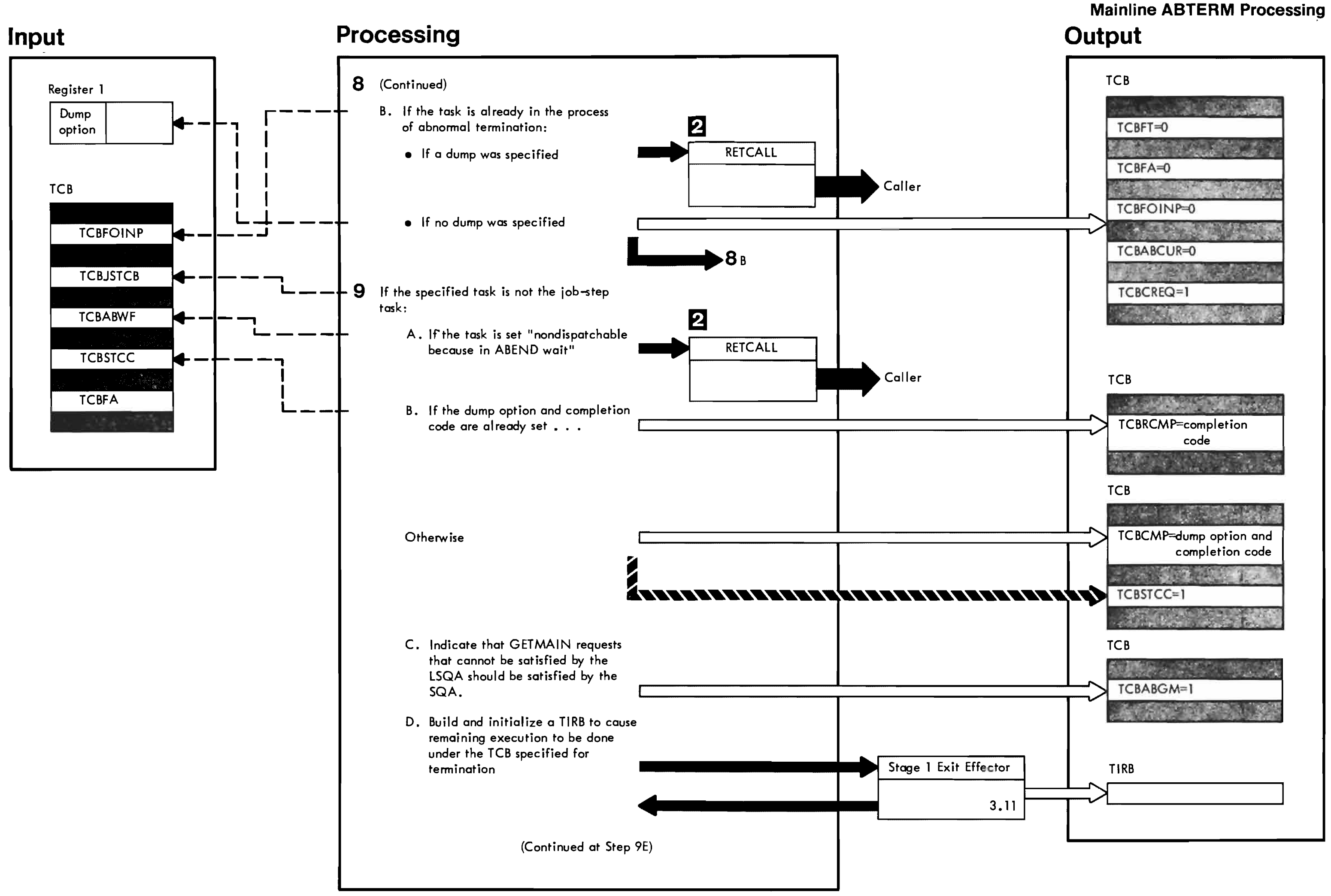
A. If the task is not already in the process of abnormal termination

(Continued at Step 8B)

Diagram 8.12 (Steps 1-8A) Mainline ABTERM Processing (Module IEAVAB00)

NOTES	ROUTINE NAME	LABEL
1 These routines enter ABTERM at IEA0AB00: External SLIH; Machine Check Handler; user type-1 SVCs; user and system type-2, 3, and 4 SVCs; system routines.	IEA0AB00	TCBTEST
2 The internal subroutine, RETCALL, returns to the caller if the return code address points neither to the Type-1 Exit routine or the dispatcher. If the return address points to either the Type-1 Exit routine or the dispatcher, RETCALL checks the type-1 switch (IEAPTYPE1) in the SVC FLIH. If the switch is set, RETCALL passes control to the Type-1 Exit routine. If it is not set, RETCALL passes control to the dispatcher.		VALIDTCB
2 CVTPGSUP contains the address of the paging supervisor. This address is compared with the specified TCB address.		
3		STAYEREC
4 The operator's CANCEL command could request ABTERM processing for a task that has completed normal EOT processing. TCBFC, when set, means the task is complete.		
5 TCBIWAIT, when set, means the task is nondispatchable due to an input wait condition. TCBOWAIT, when set, means the task is nondispatchable due to an output wait condition.		
6 TCBSTABE is the STA recursion flag. If it is set, TCBABTRM is not tested -- this task must be scheduled for ABEND.		
7 TCBABTRM is the ABTERM recursion flag set in Step 9.		
8 TCBOTC contains the address of the originating TCB. It matches the address contained in IEATCBP+4 (current TCB) if the initiator is the caller. TCBJSTCB contains the address of the job-step TCB.		ACESS1
After resetting the selected primary nondispatchability flags, the stop count is zeroed. STATUS is then called again to reset the selected secondary nondispatchability flags.		
A. TCBFA, when set, means this task is being abnormally terminated.		JSTABEND

Diagram 8.12 (Steps 8B-9D)
Mainline ABTERM Processing



8 (Continued)

B. If the task is already in the process of abnormal termination:

- If a dump was specified
- If no dump was specified

9 If the specified task is not the job-step task:

A. If the task is set "nondispatchable because in ABEND wait"

B. If the dump option and completion code are already set . . .

Otherwise

C. Indicate that GETMAIN requests that cannot be satisfied by the LSQA should be satisfied by the SQA.

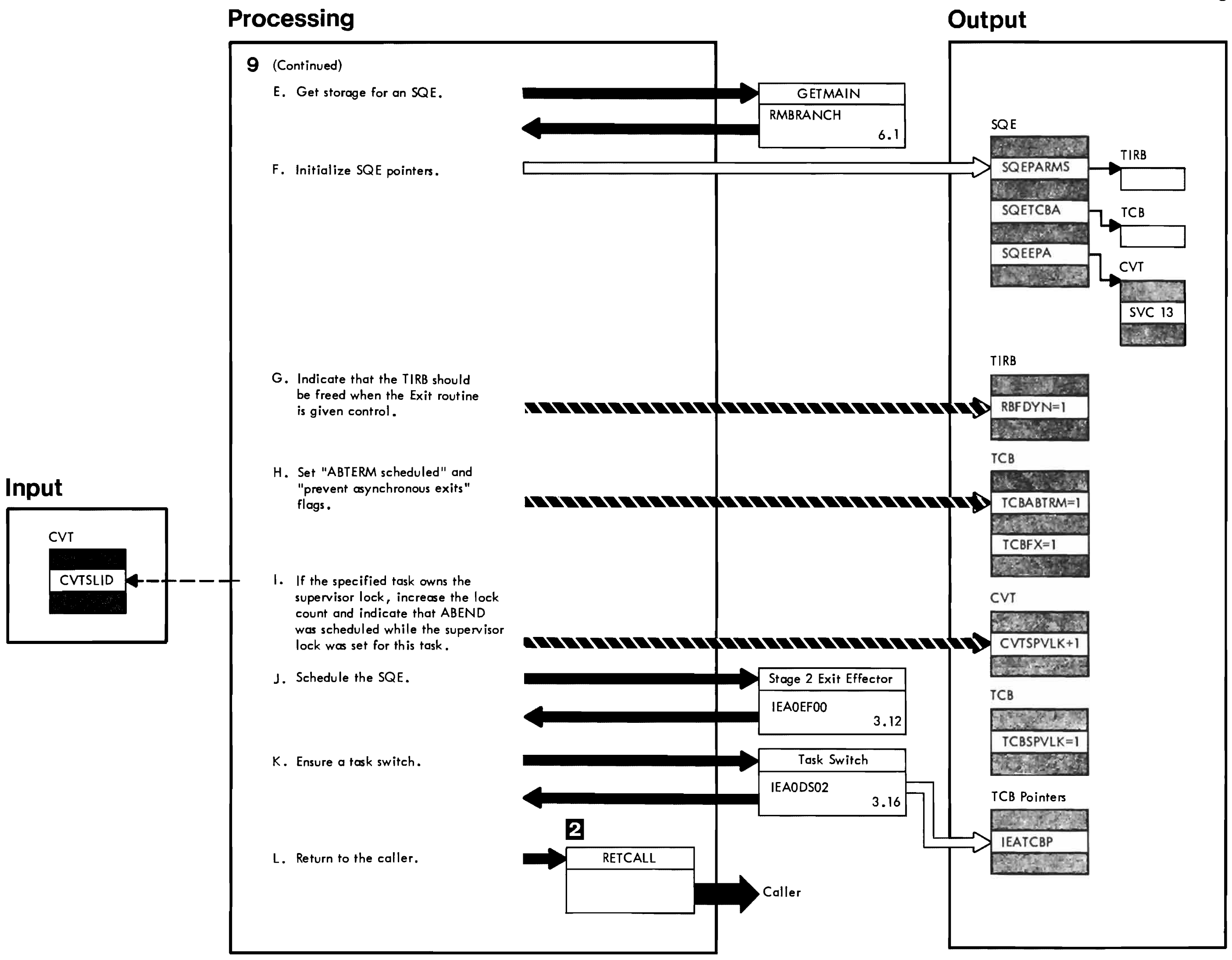
D. Build and initialize a TIRB to cause remaining execution to be done under the TCB specified for termination

(Continued at Step 9E)

Diagram 8.12 (Steps 8B-9D) Mainline ABTERM Processing (Module IEAVAB00)

NOTES	ROUTINE NAME	LABEL
8 B. The task may already be in abnormal termination if the operator issued a CANCEL command or the job-step timer expired. If no dump was specified (for example, a CANCEL command without the dump option was issued), all flags that indicate ABEND processing are cleared to give the appearance of a first-time request for termination. The stacking flag in the job-step TCB is reset if TCBFOINP is set.	IEA0AB00	
9 TCBJSTCB contains the address of the job-step TCB.		ACESS1
A. TCBABWF is set by ABEND, and means the task is in a wait state. ABEND should not be scheduled because a related higher-level task is already being abnormally terminated. Resources for this task are released as part of the termination processing for the higher-level task.		ABWAIT
B. TCBSTCC=0 means the completion code and dump flag have not yet been saved. The completion code is displayed in the dump as a part of the TCB; it is made available to the attaching task via the ABEND routine, and it is displayed in a GTF trace, if GTF is active.		SCHEDULE
C.		ABTRMGM

Diagram 8.12 (Steps 9E-9L)
Mainline ABTERM Processing



9 (Continued)

E. Get storage for an SQE.

F. Initialize SQE pointers.

G. Indicate that the TIRB should be freed when the Exit routine is given control.

H. Set "ABTERM scheduled" and "prevent asynchronous exits" flags.

I. If the specified task owns the supervisor lock, increase the lock count and indicate that ABEND was scheduled while the supervisor lock was set for this task.

J. Schedule the SQE.

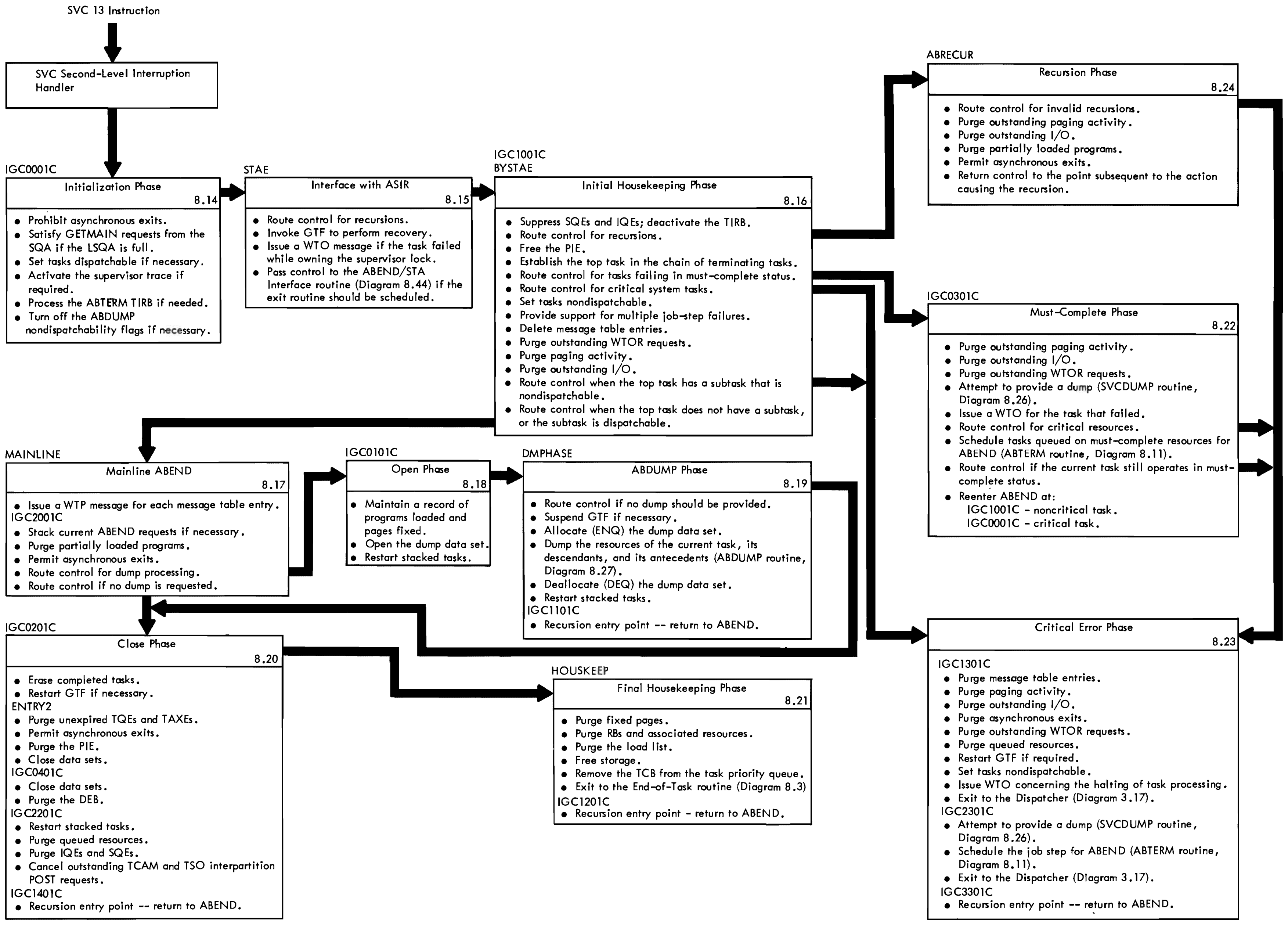
K. Ensure a task switch.

L. Return to the caller.

Diagram 8.12 (Steps 9E-9L) Mainline ABTERM Processing (Module IEA0AB00)

NOTES			ROUTINE NAME	LABEL
9	H.	TCBABTRM is checked by this routine on recursion. (See Step 7.) TCBFX is set to prevent the scheduling of a user's exit routine for the task by the Stage 3 Exit Effector during dispatcher processing.	IEA0AB00	
	I.	The task for which a lock is set was probably manipulating critical system queues. ABEND writes a message to the operator to warn that system problems may follow.		TSTLOCK
	J.			SQEINIT
	K.	The Task Switch routine determines whether a higher-priority ready task should be given control when the dispatcher is next entered. The Dispatcher routes control to the SVC 13 instruction pointed to in the SQE built in Step 9F.		

Diagram 8.13 Overview of the ABEND Routine



SVC 13 Instruction

SVC Second-Level Interruption Handler

IGC0001C
Initialization Phase 8.14
- Prohibit asynchronous exits.
- Satisfy GETMAIN requests from the SQA if the LSQA is full.
- Set tasks dispatchable if necessary.
- Activate the supervisor trace if required.
- Process the ABTERM TIRB if needed.
- Turn off the ABDUMP nondispatchability flags if necessary.

STAE
Interface with ASIR 8.15
- Route control for recursions.
- Invoke GTF to perform recovery.
- Issue a WTO message if the task failed while owning the supervisor lock.
- Pass control to the ABEND/STA Interface routine (Diagram 8.44) if the exit routine should be scheduled.

IGC1001C
BYSTAE
Initial Housekeeping Phase 8.16
- Suppress SQEs and IQEs; deactivate the TIRB.
- Route control for recursions.
- Free the PIE.
- Establish the top task in the chain of terminating tasks.
- Route control for tasks failing in must-complete status.
- Route control for critical system tasks.
- Set tasks nondispatchable.
- Provide support for multiple job-step failures.
- Delete message table entries.
- Purge outstanding WTOR requests.
- Purge paging activity.
- Purge outstanding I/O.
- Route control when the top task has a subtask that is nondispatchable.
- Route control when the top task does not have a subtask, or the subtask is dispatchable.

ABRECUR
Recursion Phase 8.24
- Route control for invalid recursions.
- Purge outstanding paging activity.
- Purge outstanding I/O.
- Purge partially loaded programs.
- Permit asynchronous exits.
- Return control to the point subsequent to the action causing the recursion.

IGC0301C
Must-Complete Phase 8.22
- Purge outstanding paging activity.
- Purge outstanding I/O.
- Purge outstanding WTOR requests.
- Attempt to provide a dump (SVCDUMP routine, Diagram 8.26).
- Issue a WTO for the task that failed.
- Route control for critical resources.
- Schedule tasks queued on must-complete resources for ABEND (ABTERM routine, Diagram 8.11).
- Route control if the current task still operates in must-complete status.
- Reenter ABEND at:
  IGC1001C - noncritical task.
  IGC0001C - critical task.

MAINLINE
Mainline ABEND 8.17
- Issue a WTP message for each message table entry.

IGC2001C
- Stack current ABEND requests if necessary.
- Purge partially loaded programs.
- Permit asynchronous exits.
- Route control for dump processing.
- Route control if no dump is requested.

IGC0101C
Open Phase 8.18
- Maintain a record of programs loaded and pages fixed.
- Open the dump data set.
- Restart stacked tasks.

DMPHASE
ABDUMP Phase 8.19
- Route control if no dump should be provided.
- Suspend GTF if necessary.
- Allocate (ENQ) the dump data set.
- Dump the resources of the current task, its descendants, and its antecedents (ABDUMP routine, Diagram 8.27).
- Deallocate (DEQ) the dump data set.
- Restart stacked tasks.

IGC1101C
- Recursion entry point -- return to ABEND.

IGC0201C
Close Phase 8.20
- Erase completed tasks.
- Restart GTF if necessary.

ENTRY2
- Purge unexpired TQEs and TAXEs.
- Permit asynchronous exits.
- Purge the PIE.
- Close data sets.

IGC0401C
- Close data sets.
- Purge the DEB.

IGC2201C
- Restart stacked tasks.
- Purge queued resources.
- Purge IQEs and SQEs.
- Cancel outstanding TCAM and TSO interpartition POST requests.

IGC1401C
- Recursion entry point -- return to ABEND.

HOUSKEEP
Final Housekeeping Phase 8.21
- Purge fixed pages.
- Purge RBs and associated resources.
- Purge the load list.
- Free storage.
- Remove the TCB from the task priority queue.
- Exit to the End-of-Task routine (Diagram 8.3)

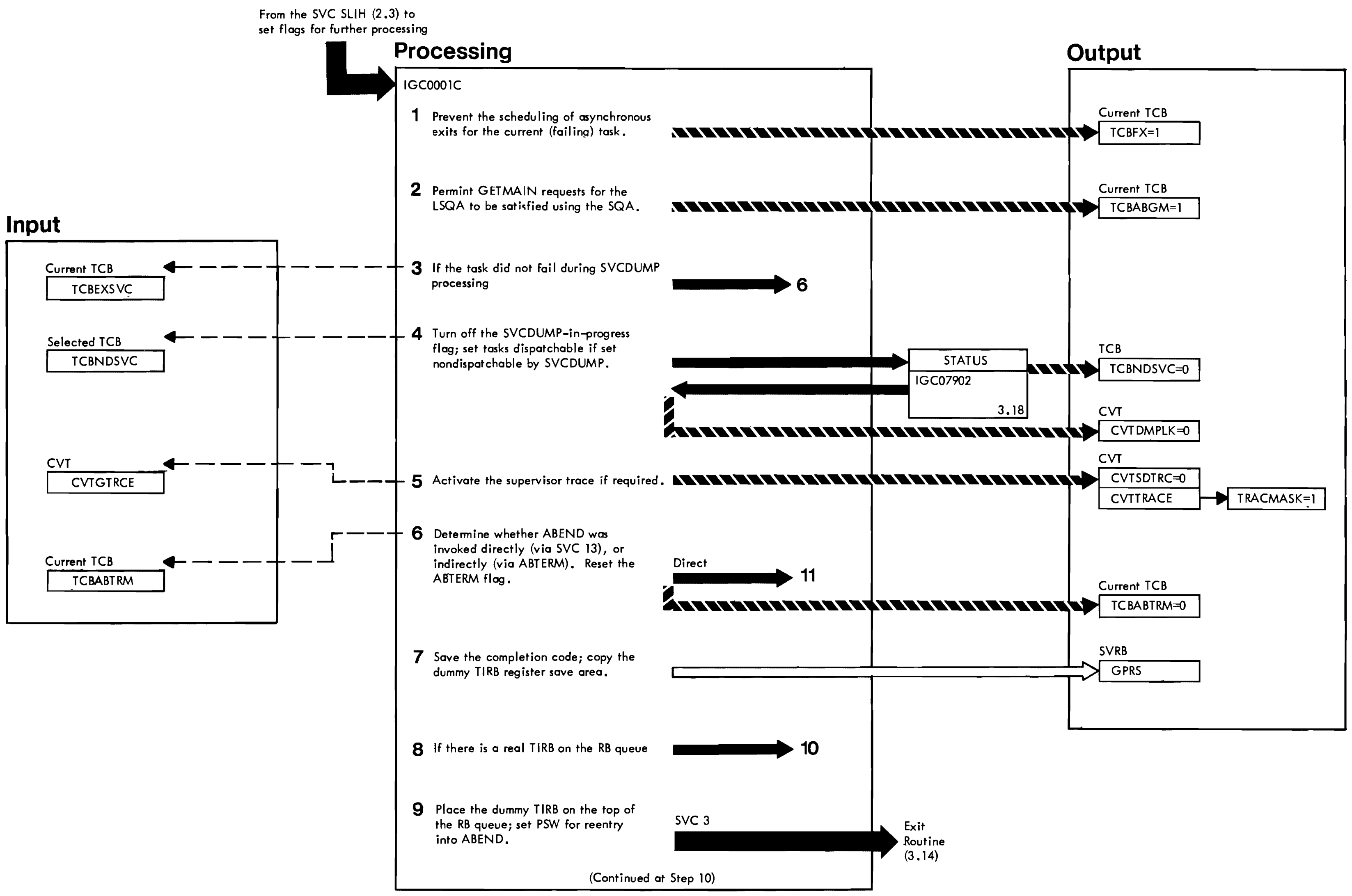
IGC1201C
- Recursion entry point - return to ABEND.

Critical Error Phase 8.23

IGC1301C
- Purge message table entries.
- Purge paging activity.
- Purge outstanding I/O.
- Purge asynchronous exits.
- Purge outstanding WTOR requests.
- Purge queued resources.
- Restart GTF if required.
- Set tasks nondispatchable.
- Issue WTO concerning the halting of task processing.
- Exit to the Dispatcher (Diagram 3.17).

IGC2301C
- Attempt to provide a dump (SVCDUMP routine, Diagram 8.26).
- Schedule the job step for ABEND (ABTERM routine, Diagram 8.11).
- Exit to the Dispatcher (Diagram 3.17).

IGC3301C
- Recursion entry point -- return to ABEND.



1 Prevent the scheduling of asynchronous exits for the current (failing) task.

2 Permint GETMAIN requests for the LSQA to be satisfied using the SQA.

3 If the task did not fail during SVCDUMP processing

4 Turn off the SVCDUMP-in-progress flag; set tasks dispatchable if set nondispatchable by SVCDUMP.

5 Activate the supervisor trace if required.

6 Determine whether ABEND was invoked directly (via SVC 13), or indirectly (via ABTERM). Reset the ABTERM flag.

7 Save the completion code; copy the dummy TIRB register save area.

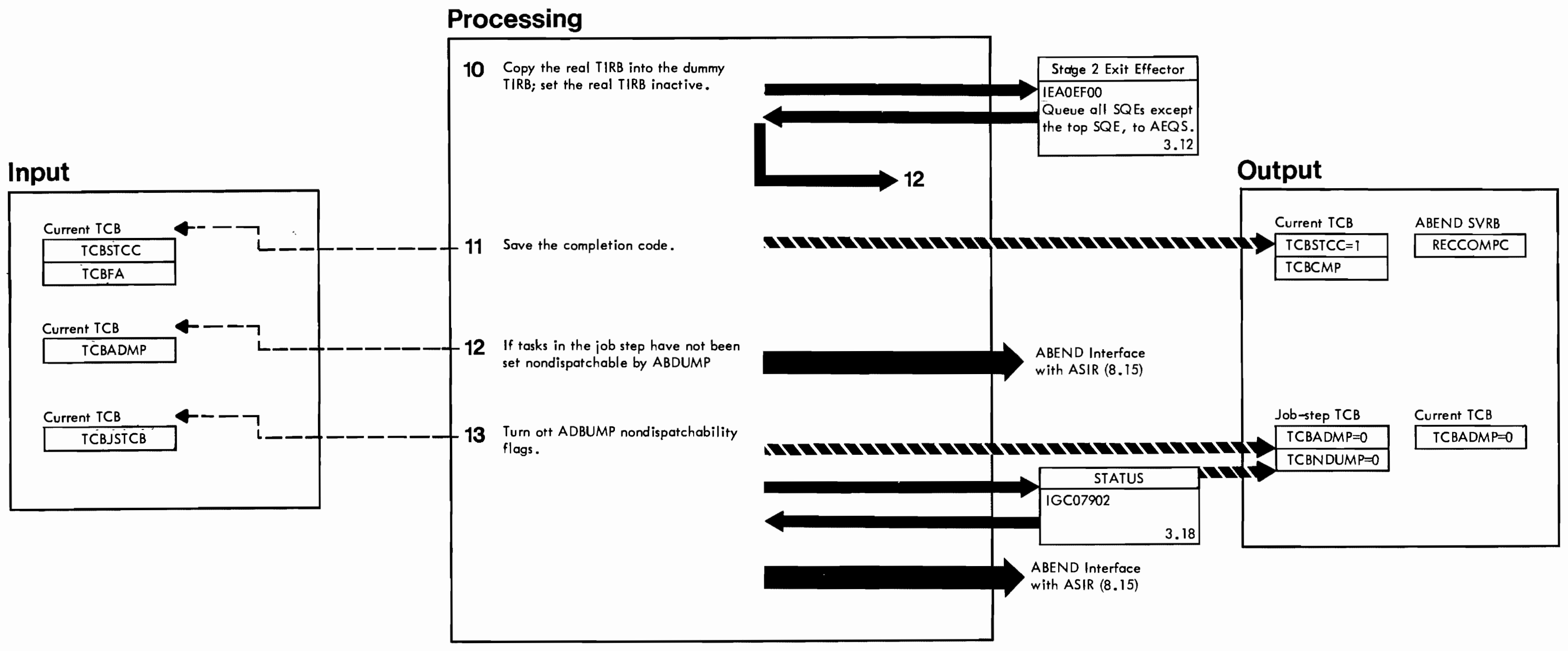
8 If there is a real TIRB on the RB queue

9 Place the dummy TIRB on the top of the RB queue; set PSW for reentry into ABEND.

(Continued at Step 10)

Diagram 8.14 (Steps 1-9) ABEND Initialization Phase (Module IEAVTM00)

NOTES		ROUTINE NAME	LABEL
1	Asynchronous exits are prohibited to avoid interruptions. The recursion code is saved and the TCBABCUR field is zeroed out.	Initialization Phase	
2	Because GETMAIN requests can be satisfied using the SQA (system queue area), recursions due to insufficient LSQA (local system queue area) storage are prevented.		
4	If tasks sharing the same LSQA as the task being dumped are nondispatchable due to SVCDUMP processing (TCBNDSVC ≠ 0), they are set dispatchable at this time. All tasks represented on the task queue are processed.		SVCDLOOP SVCDNXT
5	The SVCDUMP flag (CVTSDTRC) is turned off to indicate that the failure occurred during SVCDUMP processing. If the supervisor trace is requested to be started (CVTGTRCE = 0), the TRACMASK field is turned on.		
6			NOSVCDMP
7	The completion code saved by ABTERM is stored in the ABEND SVRB RSA. The dummy TIRB (task interruption request block) register save area is copied into the ABEND SVRB register save area.		ABTRM
8			TIRBLOOP
9	The PSW is set to the address of SETADMP (Step 12); control returns to this point after the Exit routine has completed processing. ABEND registers are saved in the TIRB register save area before exiting.		NORLTIRB



10 Copy the real TIRB into the dummy TIRB; set the real TIRB inactive.

11 Save the completion code.

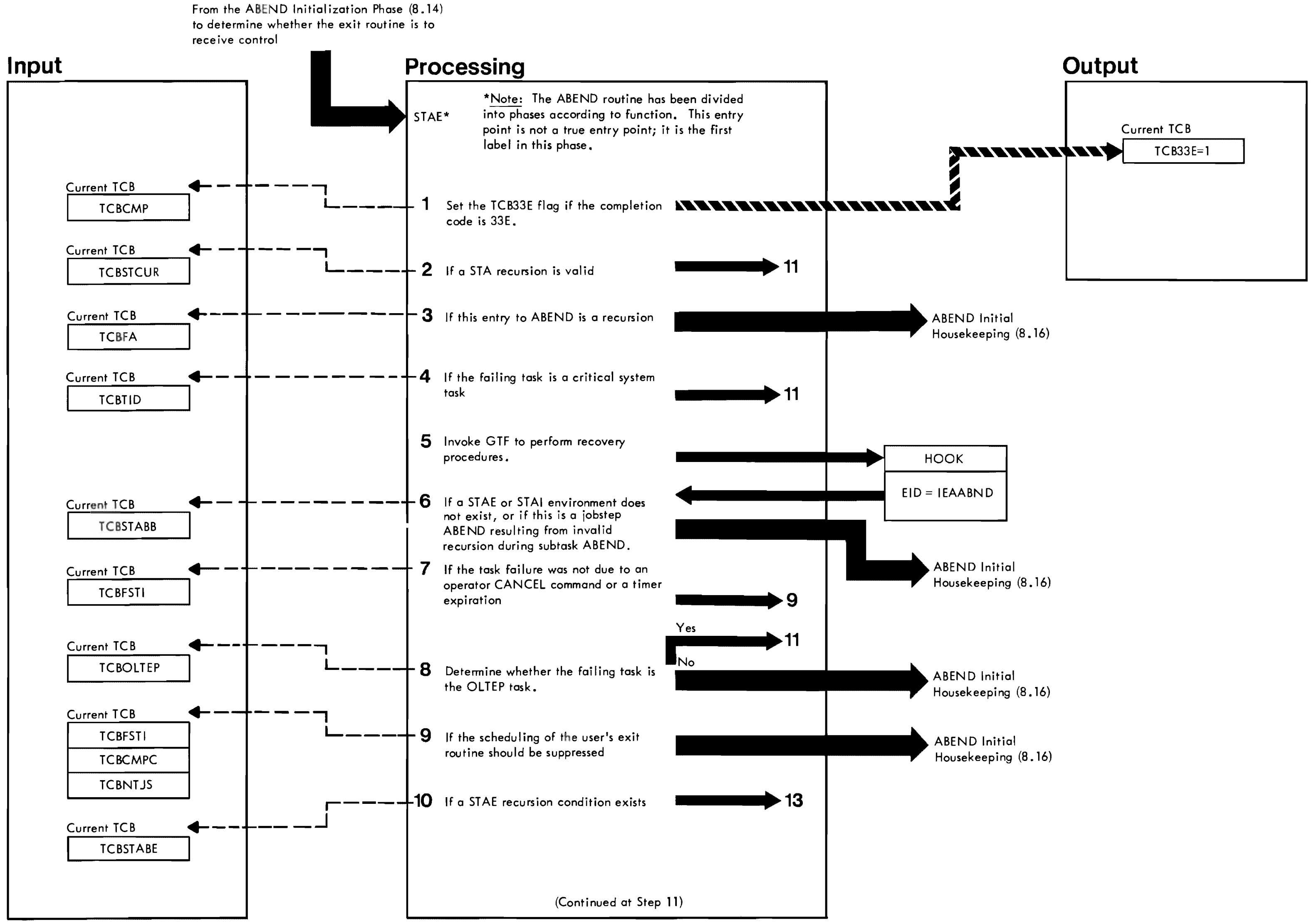
12 If tasks in the job step have not been set nondispatchable by ABDUMP

13 Turn off ADBUMP nondispatchability flags.

Diagram 8.14 (Steps 10-13) ABEND Initialization Phase (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
10 After removing the dummy TIRB from the RB queue, the real TIRB is copied into the dummy TIRB. The ABTERM TIRB is then queued onto the RB queue at the location occupied by the real TIRB, which is set inactive. The Stage 2 Exit Effector then queues SQEs (supervisor queue elements) to the AEQS (SQE asynchronous exit queue) queue.	Initial-ization Phase	REALTIRB SQELOOP
11 If this entry to ABEND is a recursion (TCBFA = 1), the completion code is saved in the ABEND SVRB. Otherwise, if the completion code is to be saved (TCBSTCC = 0), it is stored in the TCB. The TCBSTCC flag is set to indicate that the completion code in the TCB is not to be overlaid. The completion code is provided in diagnostic messages should a recursion occur.		NOABTRM RECURS
12 If the failure occurred in ABDUMP processing and tasks in the job step have not been set nondispatchable (TCBADMP = 0), execution continues with the ABEND Interface with ASIR (Diagram 8.15).		SETADMP
13 The nondispatchability flag TCBADMP is turned off in the current and job-step TCBs to avoid further entries to STATUS in the event of recursion.		

• Diagram 8.15 (Steps 1-10)
ABEND Interface with ASIR



From the ABEND Initialization Phase (8.14) to determine whether the exit routine is to receive control

Input

Current TCB: TCBCMP
Current TCB: TCBSTCUR
Current TCB: TCBFA
Current TCB: TCBTID
Current TCB: TCBSTABB
Current TCB: TCBFSTI
Current TCB: TCBOLTEP
Current TCB: TCBFSTI, TCBCMPC, TCBNTJS
Current TCB: TCBSTABE

Processing

STAE*

*Note: The ABEND routine has been divided into phases according to function. This entry point is not a true entry point; it is the first label in this phase.

1 Set the TCB33E flag if the completion code is 33E.

2 If a STA recursion is valid → 11

3 If this entry to ABEND is a recursion → ABEND Initial Housekeeping (8.16)

4 If the failing task is a critical system task → 11

5 Invoke GTF to perform recovery procedures. → HOOK, EID = IEAABND

6 If a STAE or STAI environment does not exist, or if this is a jobstep ABEND resulting from invalid recursion during subtask ABEND. → ABEND Initial Housekeeping (8.16)

7 If the task failure was not due to an operator CANCEL command or a timer expiration → 9

8 Determine whether the failing task is the OLTEP task. Yes → 11; No → ABEND Initial Housekeeping (8.16)

9 If the scheduling of the user's exit routine should be suppressed → ABEND Initial Housekeeping (8.16)

10 If a STAE recursion condition exists → 13

(Continued at Step 11)

Output

Current TCB: TCB33E=1

Diagram 8.15 (Steps 1-10) ABEND Interface with ASIR (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
1 A completion code of 33E indicates that a DETACH macro instruction, specifying the STAE=YES operand, has been issued by the originating task. However, the specified subtask has not completed execution. The TCB33E flag is used by ASIR (ABEND/STA Interface Routine).	ABEND Inter-face with ASIR	
2 The three user exits are STAE, STAI, and STAR.		TSTSTAR
4 If the failing (terminating) task is a critical system task, task ID (TCBTID) is greater than 199, ABEND assumes that a STAR exit exists. The STAR exit is for critical system tasks only; it receives control only after any valid STAE/STAI exits have been scheduled. The critical system tasks that have a STAR exit are the master scheduler task, the system error task, the communications task, and the IORMS (I/O Recovery Management Support) task.		
5 GTF (generalized trace facility) performs necessary recovery procedures if a GTF task is failing. (See <u>OS/VS Service Aids</u> for information on GTF.)		
6 If a STA control block does not exist (TCBSTABB = 0), control passes to the Initial Housekeeping Phase (Diagram 8.16).		
8 See <u>OS/VS OLTEP</u> for information on the OLTEP task.		
9 The scheduling of the user exit routine is suppressed if any of the following conditions exist:		
• The DETACH macro instruction was issued for an in-complete subtask (TCBCMPC = X'13E').		NOCANCEL
• All tasks in the job step are in a 30-minute wait (TCBCMPC = X'522').		
• The limit on the SYSOUT DD card has been exceeded (TCBCMPC = X'722').		
• The failing task, although a job-step task, is not the highest task in the chain of terminating tasks (TCBNTJS = 1).		
10		TSTRECUR

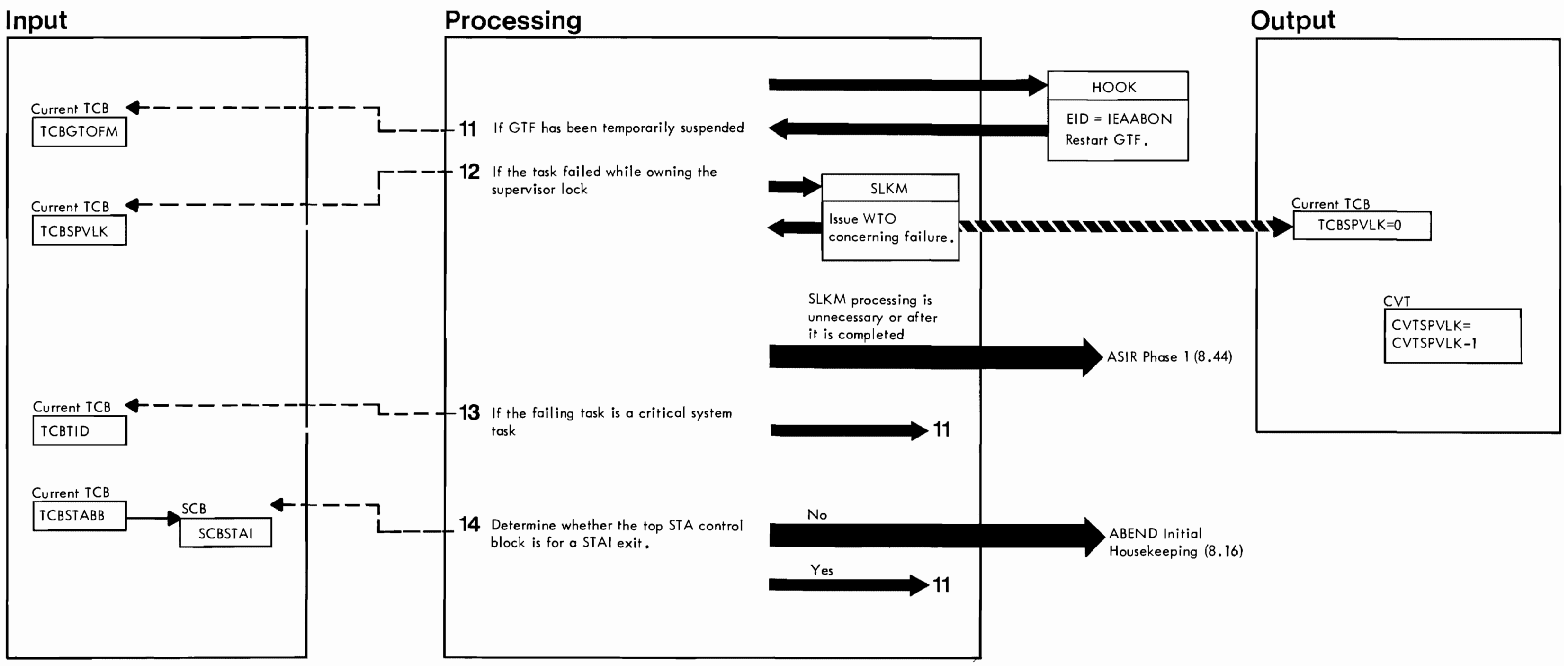
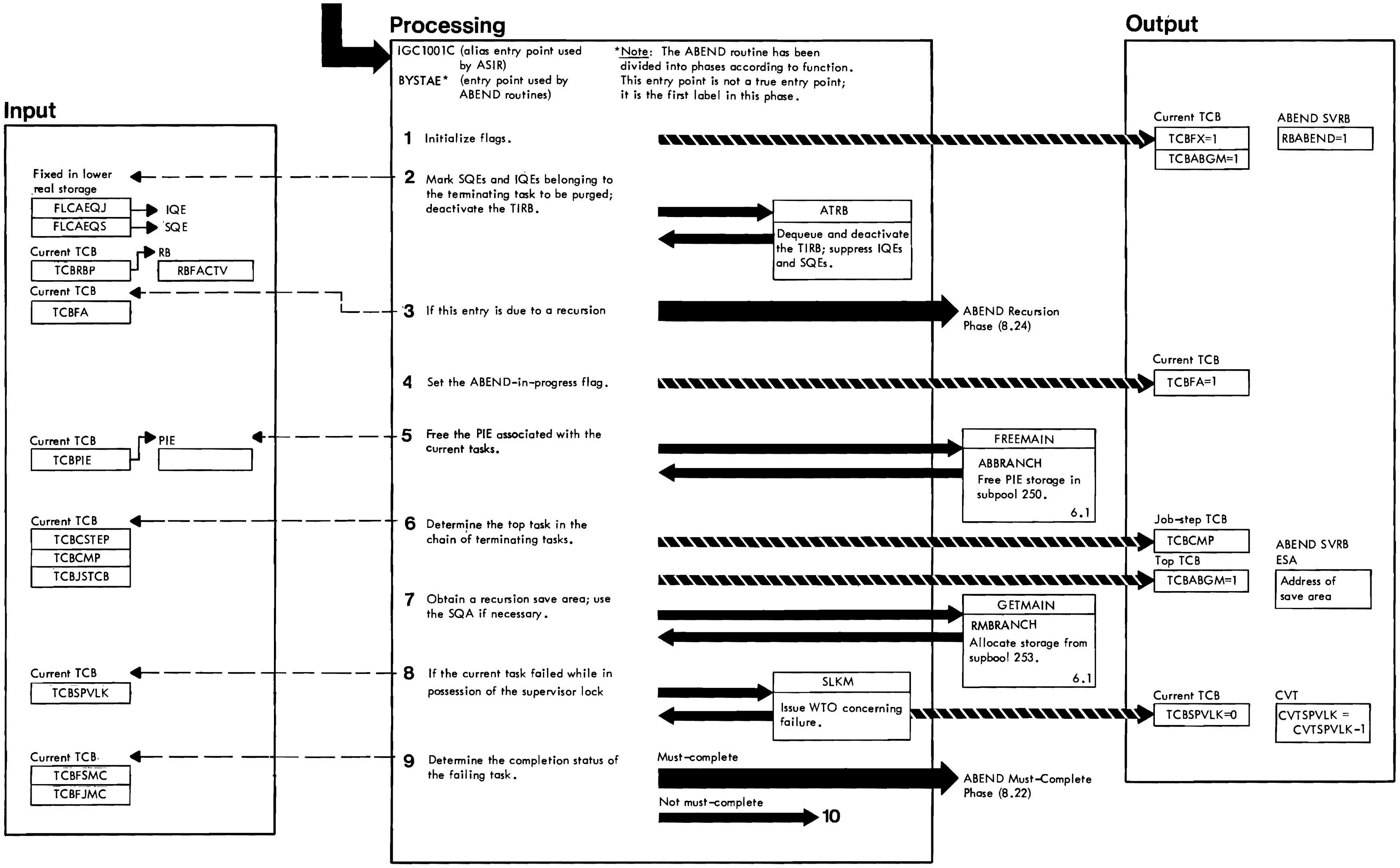

Input
Processing
Output
Current TCB
TCBGTOFM
Current TCB
TCBSPVLK
Current TCB
TCBTID
Current TCB
TCBSTABB
SCB
SCBSTAI
11 If GTF has been temporarily suspended
12 If the task failed while owning the supervisor lock
13 If the failing task is a critical system task
14 Determine whether the top STA control block is for a STAI exit.
HOOK
EID = IEAABON
Restart GTF.
SLKM
Issue WTO concerning failure.
SLKM processing is unnecessary or after it is completed
ASIR Phase 1 (8.44)
11
No
ABEND Initial Housekeeping (8.16)
Yes
11
Current TCB
TCBSPVLK=0
CVT
CVTSPVLK=
CVTSPVLK-1

Diagram 8.15 (Steps 11-14) ABEND Interface with ASIR (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
11	ABEND Inter-face with ASIR	STAEXIT
12 The supervisor lock is an indicator used to inhibit entry to disabled code while a disabled page fault is being resolved. It is set by the Program Check First-Level Interruption Handler when a disabled page fault is encountered. It is turned off by the dispatcher when the owner of the lock has been selected as the task to be dispatched. The assumption is that the owner of the lock would not be dispatchable unless the disabled page fault had been handled by the paging task. However, it is possible that ABTERM has made the owner dispatchable to schedule the task for abnormal termination. When ABTERM sets the owner of the lock dispatchable, a count of the number of tasks that have failed while owning the supervisor lock is increased in the CVT (CVTSPVLK). The TCBSPVLK flag is set to indicate to ABEND that a message is to be issued to the operator. The SLKM subroutine turns off the supervisor lock flag (TCBSPVLK), decreases the lock count (CVTSPVLK), and issues the message. (See <u>OS/VS Job Management Logic</u> for information on the WTO routine.) After SLKM processing has completed, the TCBABGM flag is turned off and control passes to ASIR.		STASLKM NCSPVLK
13		STAI
14 If the STA control block is not for a STAI exit (SCBSTAI ≠ 1), control passes to the Initial Housekeeping Phase. Otherwise, preparations are made to exit to the ABEND/ STA Interface Routine (ASIR).		

From the ABEND/STA Interface routine (8.44), the Must-Complete Phase (8.22), or the ABEND Interface with ASIR (8.15) to purge



IGC1001C (alias entry point used by ASIR)
BYSTAE* (entry point used by ABEND routines)

*Note: The ABEND routine has been divided into phases according to function. This entry point is not a true entry point; it is the first label in this phase.

1 Initialize flags.

2 Mark SQEs and IQEs belonging to the terminating task to be purged; deactivate the TIRB.

Dequeue and deactivate the TIRB; suppress IQEs and SQEs.

3 If this entry is due to a recursion

4 Set the ABEND-in-progress flag.

5 Free the PIE associated with the current tasks.

Free PIE storage in subpool 250.

6 Determine the top task in the chain of terminating tasks.

7 Obtain a recursion save area; use the SQA if necessary.

Allocate storage from supbool 253.

8 If the current task failed while in possession of the supervisor lock

Issue WTO concerning failure.

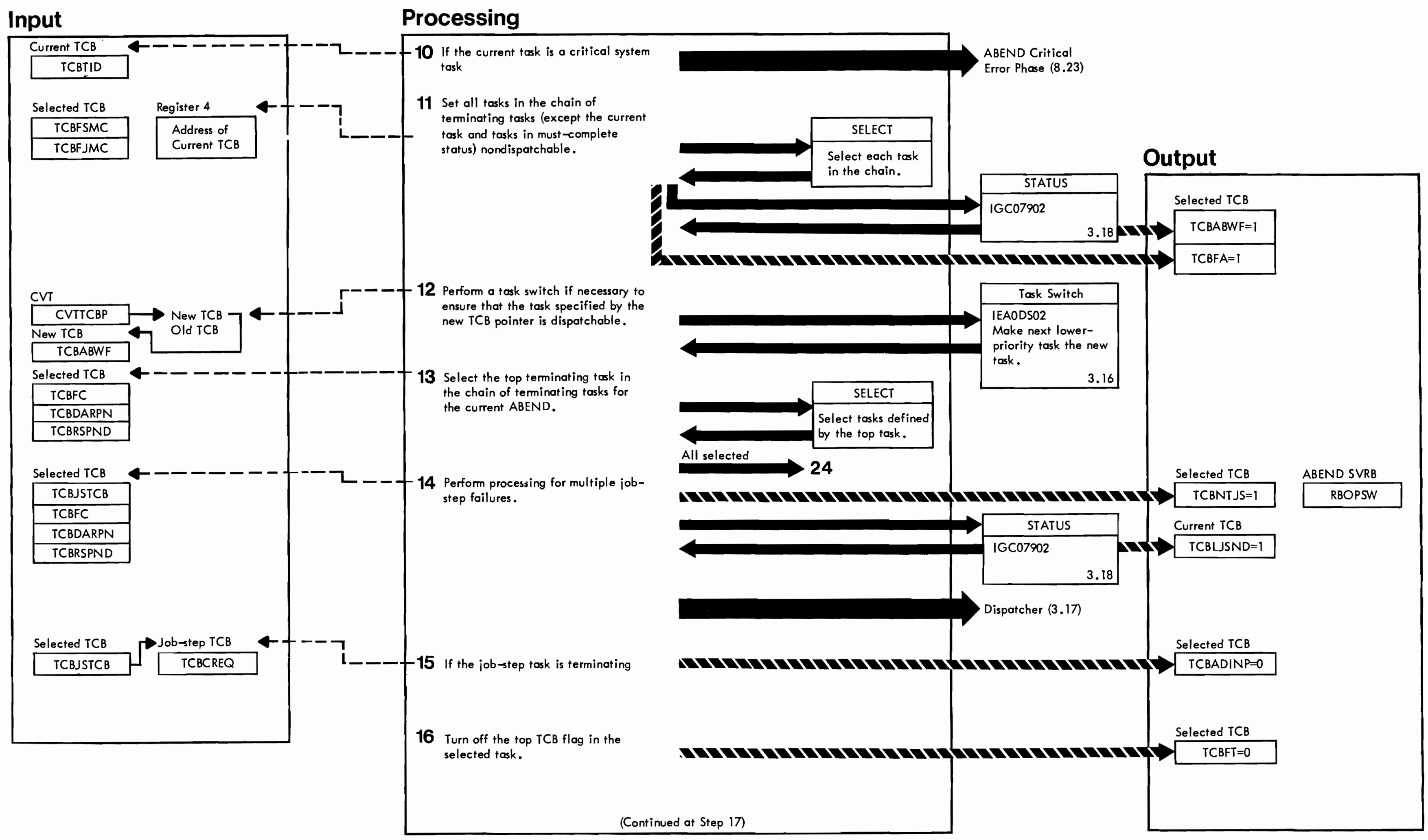
9 Determine the completion status of the failing task.

Diagram 8.16 (Steps 1-9) ABEND Initial Housekeeping Phase (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
1 The following flags are initialized for the current (failing) task: • TCBFX -- prevent asynchronous exits. • RBABEND -- identify the ABEND SVRB in case of a recursion. • TCBABGM -- set default for GETMAIN requests from the SQA if the LSQA cannot satisfy the request.	Initial House-keeping Phase	
2 The internal subroutine ATRB first marks all IQEs (interruption queue elements) in the AEQJ (IQE asynchronous exit queue) to be purged. When the AEQJ queue has been exhausted, all SQEs (supervisor queue elements) on the AEQS (SQE asynchronous exit queue) are marked to be purged. IQEs and SQEs are not purged at this time because they represent status that might be valuable if a SYSABEND dump is provided. These queue elements are dequeued and the associated storage freed after any attempts to provide a dump have been made. Upon completion of SQE processing, the TIRB (task interruption request block) is tested to see if it is active. If the TIRB is not active (RBFACTV ≠ 1), control returns to the caller. Otherwise, any SQEs chained to the TIRB are marked to be purged, dequeued from the TIRB SQE queue, and sent to the Stage 2 Exit Effector to be requeued on the AEQS. When all SQEs have been processed, the TIRB is dequeued from the TCB RB queue. The TIRB is deactivated to avert an interlock. The TIRB is a serially reusable resource which might currently be in use; the entry to ABEND has made normal completion of its use impossible.	ATRB ATRB	ATRBMIQE ATRBSQES ATRBTRTN ATRBEND ATRBSPGE ATRBRDQ
5 If a PIE exists for the current task, it is freed so that a user SPIE exit routine does not gain control if a program check is encountered during ABEND. A special branch entry (ABBRANCH) into FREEMAIN is used to prevent a recursion, which could result because the PIE is allocated from user storage.	Initial House-keeping Phase	

NOTES	ROUTINE NAME	LABEL
6 The top terminating task is the highest-level task in the chain of terminating tasks. If an ABEND macro instruction was specified with the STEP option (TCBCSTEP=1), the top terminating task in the chain is defined by the job-step task (TCBJSTCB); all tasks in the job step are terminated. The completion code is moved from the current TCB into the job-step TCB. If the STEP option has not been specified, the top terminating task is defined by the current (failing) task; all incomplete descendants of this task are terminated. The completion code remains in the current TCB.	Initial House-keeping Phase	TSTSTEP STEPOPT
7 The top terminating task is used to allocate storage since storage is requested from subpool 253. If the current task is used, and a STEP option has been specified, the save area would be freed prematurely when storage belonging to each task is purged. The top terminating task is processed last and its storage cannot be freed early.		GETRECSA
8 If the supervisor lock is owned by the failing task, the registers are saved in the recursion save area and the recursion flag (TCBABCUR = X'01') is set during SLKM processing. The recursion flag is cleared upon return from SLKM.		SUPLKCHK MCMPTEST
9 If the task failed while operating in either system-must-complete (TCBFSMC = 1), or step-must-complete (TCBFJMC = 1) status, processing continues with the Must-Complete Phase. A task is in must-complete status because it owns critical resources. Other tasks in the system or job step are set nondispatchable while the must-complete task completes processing and frees the critical resources.		MCTEST

Diagram 8.16 (Steps 10-16)
ABEND Initial Housekeeping Phase



10 If the current task is a critical system task

11 Set all tasks in the chain of terminating tasks (except the current task and tasks in must-complete status) nondispatchable.

12 Perform a task switch if necessary to ensure that the task specified by the new TCB pointer is dispatchable.

13 Select the top terminating task in the chain of terminating tasks for the current ABEND.

14 Perform processing for multiple job-step failures.

15 If the job-step task is terminating

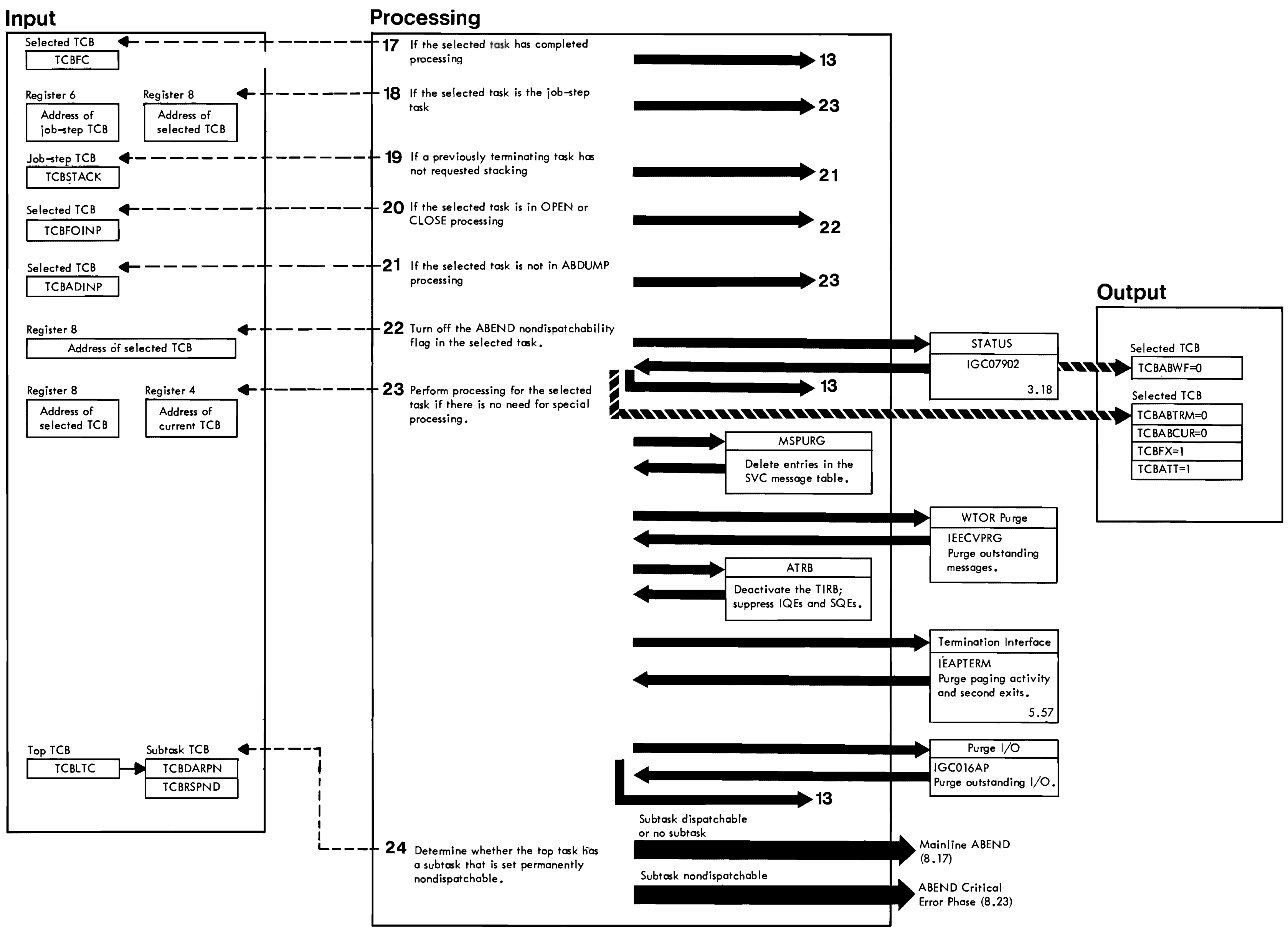
16 Turn off the top TCB flag in the selected task.

(Continued at Step 17)

Diagram 8.16 (Steps 10-16) ABEND Initial Housekeeping Phase (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
10 The TCB for a critical system task cannot be erased.	Initial House-keeping Phase	SYSTASK
11 The top terminating task is used to find all tasks in the chain. The initial input to the SELECT subroutine is the complement (negative) address of the TCB representing the top terminating task. The SELECT subroutine makes the input address positive and returns to the caller. (The address of the top terminating task's TCB is assumed to be in the ABEND SVRB ESA.) Subsequent entries to the SELECT subroutine require, as input, the address of the last TCB selected. The following order is used to find each TCB after the first one:	SELECT	SELOOP1 SELFIRST SELRET
1. If the task represented by the input TCB has a subtask (TCBLTC), the output is the address of the subtask's TCB.		SELTEST1 SELSUB
2. If the input TCB address is that of the TCB representing the top terminating task, all tasks in the chain of terminating tasks have been selected, and the output is an address of 0.		SELLOOP SELLAST
3. If the task represented by the input TCB has a same-level task (TCBNTC), the output is the address of the same-level task's TCB.		SELTEST2 SELSIS
4. The input consists of the originating task (TCBOTC) of the task represented by the input TCB. Execution then continues with the test for the top terminating task's TCB (point 2).		SELMOM
All eligible tasks are set nondispatchable to prevent the task from using a resource that may be taken by ABEND.		
12 The new TCB pointer contains the address of the highest-priority TCB; the task represented by this TCB is dispatched next. Because previous processing may have set a task specified as new nondispatchable, ABEND must ensure that the new task is dispatchable. The Task Switch routine is called to transform the next lower-priority task into the new task.	Initial House-keeping Phase	NEWTEST NEWLOOP

NOTES	ROUTINE NAME	LABEL
13 The top terminating task found earlier is used to find all tasks in the chain of terminating tasks; the tasks are located in a top-down fashion. If the selected task has completed processing (TCBFC =1), or is nondispatchable, it is ignored and the next task is selected.	Initial House-keeping Phase	SETUPSLC TSTCOMPL
14 If the selected task is a job-step task other than the top terminating task, processing is required to handle multiple job-step failures. The lowest-level job-step task is terminated first. The SELECT subroutine is called to find all job-step tasks lower than the top job-step task. This processing is only performed if the task is neither complete nor nondispatchable:		
1. An interface is established with ABTERM (IEA0AB00, Diagram 8.12) to terminate the selected task (using SVC 13) with a completion code of X'10D' and no dump.		
2. The TCBNTJS flag is set in the selected TCB to indicate that a higher-level job-step task is awaiting the abnormal termination of the selected task.		
3. The current ABEND SVRB is set to point to location TSTSTDSP (Step 24).		
4. The current task is set temporarily nondispatchable (using STATUS).		
5. The registers are saved in the current TCB register save area.		DISPATCH
6. A task switch is ensured by setting the new TCB pointer to 0 if it is the same as the old TCB pointer.		TASKSW
7. Exit is made to the dispatcher (Diagram 3.17), which passes control to the selected task. The selected task controls the abnormal termination of itself and its subtasks.		DISPEXIT
15 If the job-step task is terminating and a dump has not been requested (TCBCREQ ≠ 1), the ABDUMP-in-progress flag (TCBADINP) is turned off. This processing is necessary because a request for the abnormal termination of a task is stacked if it has a subtask that is providing an abnormal termination dump. The TCBADINP flag setting allows the operator to issue a CANCEL command for a job-step task while its subtask is in ABDUMP processing		INITFLGS
16 The top TCB flag is turned off because the current TCB may have subtasks that are currently abnormally terminating.		ERASETOP



17 If the selected task has completed processing

18 If the selected task is the job-step task

19 If a previously terminating task has not requested stacking

20 If the selected task is in OPEN or CLOSE processing

21 If the selected task is not in ABDUMP processing

22 Turn off the ABEND nondispatchability flag in the selected task.

23 Perform processing for the selected task if there is no need for special processing.

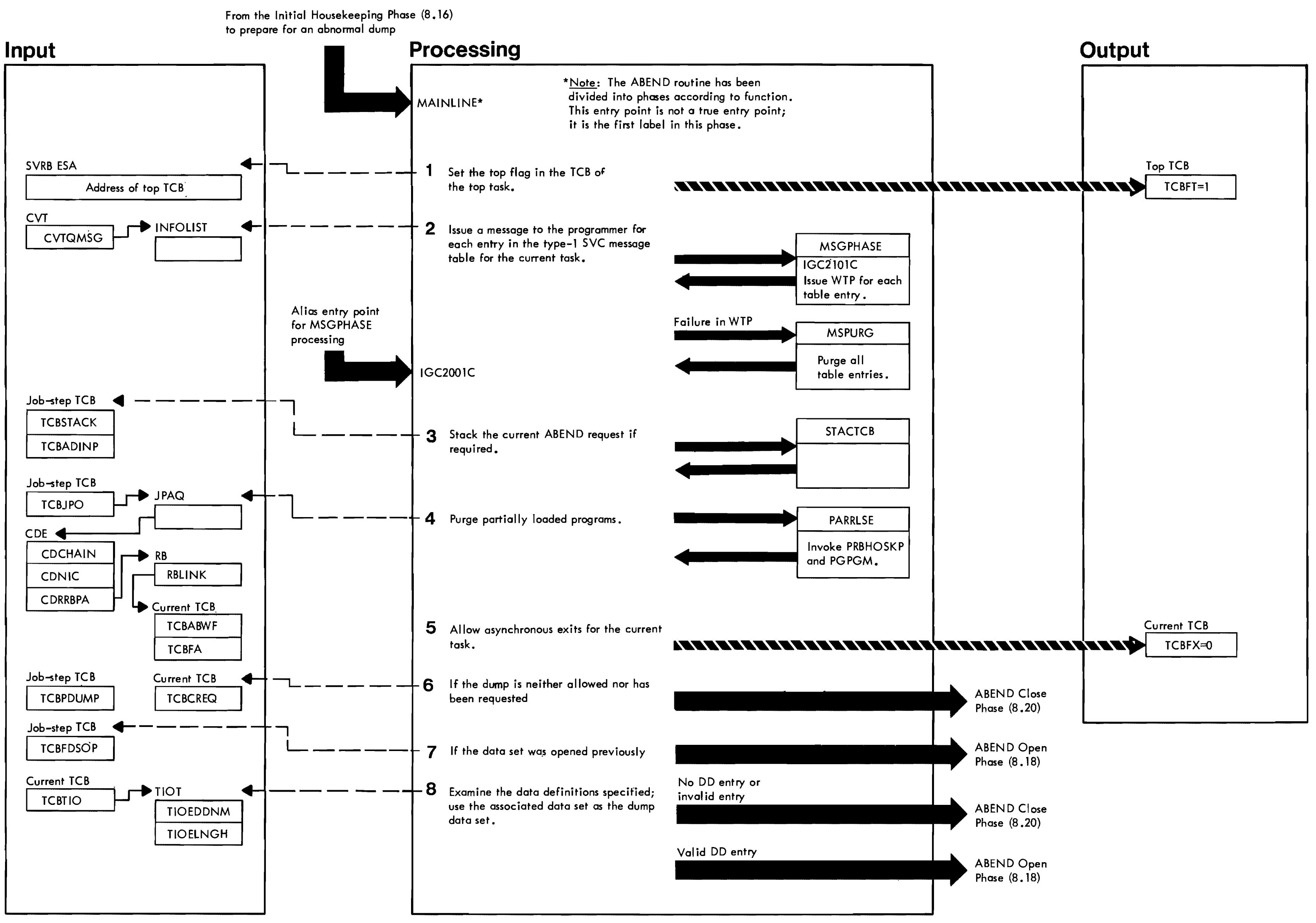
24 Determine whether the top task has a subtask that is set permanently nondispatchable.

Diagram 8.16 (Steps 17-24) ABEND Initial Housekeeping Phase (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
18 If the selected task is a job-step task, the TCBSTACK, TCBFOINP, and TCBADINP flags are not examined. TCBADINP is always set in the job-step TCB if set in a subtask TCB; the TCBSTACK flag is set to indicate that a higher-level task cannot terminate until a lower-level task completes termination processing. Therefore, it is necessary to purge the resources specified in Step 23 even though one of these flags may be set.	Initial House-keeping Phase	OCINPTST
19 If stacking has been requested, the current ABEND request is stacked until the OPEN or CLOSE processing completes. After determining that stacking is required, the STACTCB subroutine performs the following processing:	STACTCB	STACTEST
1. Modifies the ABEND SVRB old PSW so that it points to the section in the STACTCB subroutine that tests for stacking (label STACTEST).		STACKIT
2. Issues the STATUS SVC instruction to set the current task temporarily nondispatchable (TCBOCAND = 1).		
3. Ensures a task switch by setting the new TCB point to 0 if it is equal to the old TCB pointer.		STACDISP
4. Saves registers in the current TCB and exits to the Dispatcher (diagram 3.17).		STACOUT
The current ABEND request must be deferred when a dump data set is being opened. Only one OPEN macro instruction is issued for the dump data set, which remains open for the duration of the job step. If a CLOSE request has been issued, the current ABEND request must be stacked to avoid suspending CLOSE processing. The dump data set is closed only when the job-step task terminates.		
21	Initial House-keeping Phase	ADINPTST
22 If a subtask is in OPEN, CLOSE, or ABDUMP processing, the nondispatchability flag is turned off and the next task is selected. No further processing can be performed on the selected task until OPEN, CLOSE, or ABDUMP processing has completed.		ABWFOFF

NOTES	ROUTINE NAME	LABEL
23 The following functions are performed for the selected task:	Initial House-keeping Phase	
1. The ABTERM flag to prevent multiple ABEND conditions is turned off (TCBABTRM = 0).		SETFLGS
2. The valid recursion flag is cleared (TCBABCUR = 0).		
3. The scheduling of asynchronous exits is prevented (both TCBFX and TCBATT set to 1).		
4. The MSPURG subroutine is called for each task other than the current task. This subroutine erases entries in the type-1 SVC message table associated with the selected task.		
5. The WTOR Purge routine is called to purge outstanding messages associated with the selected task. (See OS/VS2 Job Management Logic for information on the WTOR Purge routine.)		PGWTOR
6. The ATRB subroutine is called to suppress IQEs and SQEs, and deactivate the TIRB associated with the selected task.		
7. The paging Termination Interface routine is called to purge all paging activity and second exits for the selected task. Second exits represent asynchronous processing that is performed as a result of the completion of paging activity required because of a FIX or LOAD request.		
8. The resident Purge I/O routine is called to purge outstanding I/O for the selected task. (See OS/VS I/O Supervisor Logic for information on the Purge I/O routine.)		
After the processing has completed, the SELECT subroutine is reentered to select the next task. The same processing (Steps 14-23) is performed for the next task.		SELOOP
24 After all tasks have been selected, the top terminating task is tested. A subtask of the top terminating task may have been set nondispatchable in the following situation: Recovery Management Support (RMS) set a task permanently nondispatchable due to a machine error.		TSTSTDSP

Diagram 8.17
Mainline ABEND Processing



*Note: The ABEND routine has been divided into phases according to function. This entry point is not a true entry point; it is the first label in this phase.

1 Set the top flag in the TCB of the top task.

2 Issue a message to the programmer for each entry in the type-1 SVC message table for the current task.

3 Stack the current ABEND request if required.

4 Purge partially loaded programs.

5 Allow asynchronous exits for the current task.

6 If the dump is neither allowed nor has been requested

7 If the data set was opened previously

8 Examine the data definitions specified; use the associated data set as the dump data set.

Diagram 8.17 Mainline ABEND Processing (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
2 The message table contains entries, supplied by a type-1 SVC routine, indicating a failure in the SVC routine. The MSGPHASE subroutine performs the following processing for each entry that is for the current task:	Mainline ABEND Process-ing	
1. Sets a valid recursion flag (TCBABCUR = X'15').		
2. Obtains storage for a message buffer (from subpool 253) if necessary.		
3. Converts the completion code in the message table into EBCDIC and moves it into the message buffer.	MSGPHASE	MSGFORM
4. Moves the message into the buffer and ensures that the message ID is correct.		
5. Converts the reason code (if specified) into EBCDIC and moves it into the buffer.		
6. Moves the job name and step name into the buffer.		MSGJOBN
7. Converts the branch entry return address (if applicable) into EBCDIC and moves it into the buffer.		
8. Converts the flag field and variable data into EBCDIC and moves them into the buffer.		MSGFLG
9. Computes the message length and issues the WTO macro instruction.		MSGLNGTH
10. Clears the valid recursion flag and zeros the message table entry.		
After all entries have been processed, ABEND is reentered at IGC2001C.		MSGALL
If the WTO processing fails, a restart routine is enter-ed at label RECT1WTP; the MSPURG subroutine is called to purge all entries in the message table for the current task. Processing then continues with Step 3. <u>NOTE:</u> The MSGPHASE subroutine is located in module IEAVTM01.	Mainline ABEND Process-ing	
3 If stacking is required, certain processing already initiated by a subtask (such as OPEN, CLOSE, or ABDUMP processing) must be completed before filling the current ABEND request.		IGC2001C
4 All partially loaded programs must be purged because of possible interlocks. An interlock might occur if later ABEND processing requested that a program be loaded which contents supervision queues indicate was already being loaded. Contents supervision would queue the ABEND request and await the completion of the loading process. However, the program would never be loaded because ABEND has already purged the I/O required to load the program. After examining the CDEs (contents directory entries) associated with the job pack area queue, PARRLSE performs the following functions for programs that are not in storage:	PARRLSE	RLSEPGM PARRLOOP

NOTES	ROUTINE NAME	LABEL
1. Invokes the PGPGM subroutine if the CDE indicates that there is not a related RB (CDRRBPA = 0). The PGPGM subroutine first determines whether an extent list has been formed; if so, PGPGM frees storage occupied by the program and extent lists from subpool 251 or 252. The CDE is then dequeued from the job pack area queue. If the CDE points to a RB, the RBCDE field is set to 0. The CDE is then freed from subpool 255. This processing is continued until all CDEs have been freed.	PARRLSE PGPGM	PARRPGM FREEPGM PGPGMFXL PGPGMDQ PGPGMFRC PGPFCDE PGPGMET
2. Invokes the PRBHOSKP subroutine to restart tasks that are awaiting the completion of the loading process. PRBHOSKP is invoked if either of the following conditions exist: (a) the RB associated with the CDE is queued to the current task's TCB, or (b) the CDE is not associated with the current task, but the task with which it is associated is in an ABEND wait state and in the process of being abnormally terminated. (It is assumed that the I/O operations for the associated task have been purged so that any tasks awaiting the loading of the program must be restarted.) The PRBHOSKP subroutine processes all RBs on the wait queue. It first sets the wait count (RBWCF) to 0 and points the RB old PSW to entry point CDCONTRL (contents supervision). The RB is then dequeued from the wait queue and the RBCDE and RBPGMQ fields are set to 0. The CDE use count (CDUSE) is decremented, and the TCB associated with the RB is located and a task switch is performed if necessary. The next RB is then processed. When all RBs have been processed, control is returned to PARRLSE, which invokes PGPGM.	PARRLSE PRBHOSKP PARRLSE	PARRHSKP PARRTCBL PRBHLOOP PREHTCBL PARRPGPM
5 Asynchronous exits are allowed because system services require their use.	Mainline ABEND Process-ing	
7 The dump data set may have been opened earlier as a result of a previous entry into ABEND by a subtask in the same job step as the current (failing) task.		
8 ABEND scans the TIOT (task I/O table) and uses the last SYSABEND or SYSUDUMP DD entry as the data set definit-tion for the dump data set. (In a TSO environment, the the user cannot be certain of what type of dump he receives because dynamic allocation uses the TIOT entries in a random fashion). If a DD entry is not found, or a dummy DD entry or invalid length is specified, control passes to the Close Phase (Diagram 8.20).		TIOTDD

Diagram 8.18
ABEND Open Phase

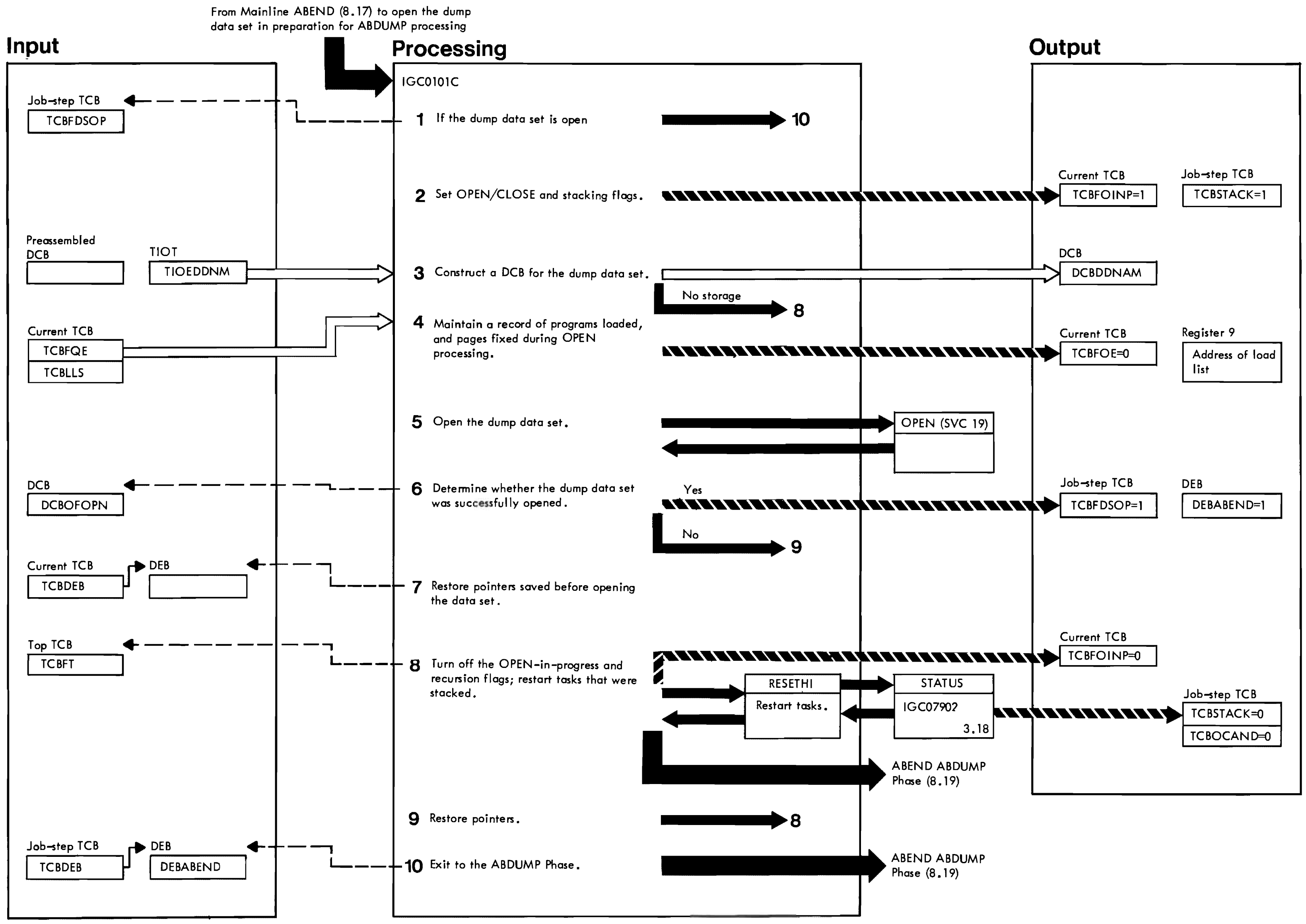

From Mainline ABEND (8.17) to open the dump data set in preparation for ABDUMP processing
Input
Processing
Output
IGC0101C
Job-step TCB
TCBFDSOP
Preassembled DCB
TIOT
TIOEDDNM
Current TCB
TCBFQE
TCBLLS
DCB
DCBOFOPN
Current TCB
TCBDEB
DEB
Top TCB
TCBFT
Job-step TCB
TCBDEB
DEB
DEBABEND
1 If the dump data set is open
10
2 Set OPEN/CLOSE and stacking flags.
3 Construct a DCB for the dump data set.
No storage
8
4 Maintain a record of programs loaded, and pages fixed during OPEN processing.
5 Open the dump data set.
OPEN (SVC 19)
6 Determine whether the dump data set was successfully opened.
Yes
No
9
7 Restore pointers saved before opening the data set.
8 Turn off the OPEN-in-progress and recursion flags; restart tasks that were stacked.
RESETHI
Restart tasks.
STATUS
IGC0790?
3.18
ABEND ABDUMP Phase (8.19)
9 Restore pointers.
8
10 Exit to the ABDUMP Phase.
ABEND ABDUMP Phase (8.19)
Current TCB
TCBFOINP=1
Job-step TCB
TCBSTACK=1
DCB
DCBDDNAM
Current TCB
TCBFOE=0
Register 9
Address of load list
Job-step TCB
TCBFDSOP=1
DEB
DEBABEND=1
Current TCB
TCBFOINP=0
Job-step TCB
TCBSTACK=0
TCBOCAND=0

Diagram 8.18 ABEND Open Phase (Module IEAVTM01)

NOTES	ROUTINE NAME	LABEL
1	Open Phase	OPHASE
2 The OPEN/CLOSE flag is set to ensure that I/O associated with OPEN processing does not get purged as a result of the failure of the originating task. Subsequent entries to ABEND are stacked until OPEN processing has completed.		
3 Before constructing a DCB, the following processing is performed if the current task is not the job-step task. A GETMAIN macro instruction is issued, immediately followed by a FREEMAIN macro instruction. This ensures that there is an SPQE (subpool queue element) fof subpool 250. The job-step TCBMSSB field is updated. A GETMAIN request is issued for 88 bytes of storage in subpool 250. If there is none available, the DCB pointer is set to indicate there will be no dump. Execution continues with Step 8. If storage has been allocated, a preassembled DCB and the appropriate DD name are moved into the allocated area. An internal switch is set to indicate the type of data set - SYSABEND or SYSUDUMP. The system must be enabled during this move.		GETDCB NODMPSW
4 Because the dump data set remains open for the duration of the job step, the associated control blocks must be queued to the job-step task so they are not purged when a subtask terminates. Any modules loaded by OPEN must remain open for the duration of the job step. A record of all pages fixed by OPEN (FIX ownership elements) is also saved.		
<u>NOTE:</u> This processing is unnecessary if the current task is the job-step task.		
5 Before opening the data set, a valid recursion flag (TCBABCUR = X'11') is set and registers are saved in the recursion save area. The valid recursion flag is cleared up on completion of OPEN processing. (See <u>OS/VS OPEN/CLOSE/EOV Logic</u> for information on the Open routine.)		JSOPEN
6 This test is made while the system is enabled. If the data set was opened, flags are set to indicate that fact and to identify the dump data set. ABEND also sets a flag (TCBFABOP) in the job-step TCB to indicate if the dump is a SYSABEND or SYSUDUMP dump. This flag is used to determine the format of additional ABEND dumps.		
7 The following processing is performed if the current task is not the job-step task:		
1. The TCBMSSB field is restored in the current task's TCB.		
2. Subroutine REQLLE is called to ensure that new programs are queued to the job-step task's load list.		
3. The dump data set is dequeued from the current TCB and queued onto the job-step TCB.		
4. The FIX Ownership Element (FOE) Merge routine (Diagram 5.26) of the paging supervisor is called to merge pages fixed during OPEN processing with pages currently fixed by the job-step task.		
5. The paging supervisor FIX Purge routine (Diagram 5.58) is called to free FOEs queued to the current task.		
6. The original FOE pointer (TCBFOE) is restored to the current TCB.		
8 After turning off the nondispatchability flag TCBOCAND (using STATUS), RESETHI examines the TCBFT flag in the top terminating task's TCB to determine whether a higher-level task failed and was stacked by ABEND (TCBFT=0). RESETHI sets the current task nondispatchable (STATUS sets the TCBABWF flag on), ensures a task switch if necessary, and passes control the the dispatcher to effect a restart at the point where processing was suspended. If the TCBFT flag is set, a higher-level task was not stacked. However, if a higher-priority task in the same job step as the current task had been stacked, a task switch could have been indicated as a result of the interface to the STATUS routine. The ABEND SVRB is set to point to the code in the RESETHI subroutine (label RSETST) which tests for a task switch. Control then passes to the dispatcher, which passes control to the higher-priority task.	RESETHI	OPRESTRT • RSETOUT
9 The valid recursion flag is cleared (TCBABCUR=0). If the data set was not opened, pointers must be restored before informing the programmer of the error. When the current task is the job-step task, these pointers are not restored. The functions consist of the following:	Open Phase	FIXLLS
1. FOEs queued to the current task are freed by the paging supervisor FIX Purge routine.		
2. The FOE pointer (TCBFOE) is restored to the current TCB.		
3. New and old programs are queued to the current task's load list by REQLLE.		
A buffer is obtained and a message is constructed informing the programmer of the failure to open the data set. A valid recursion flag (TCBABCUR = X'13') is set during WTO processing. After the message has been written, the valid recursion flag is cleared (TCBABCUR=0), the buffer is freed and the DCB pointer is cleared to indicate that no dump is to be provided. Any tasks that were stacked are now restarted (Step 8).		OPENMSG BYOPMSG NODMPSW
10 If the DEB cannot be located, control enters the ABDUMP Phase at label SETPDMP (Diagram 8.19, Step 10). Otherwise, the ABDUMP Phase is entered at DMPHASE with the address of the dump data set DCB provided.		DSOPEN

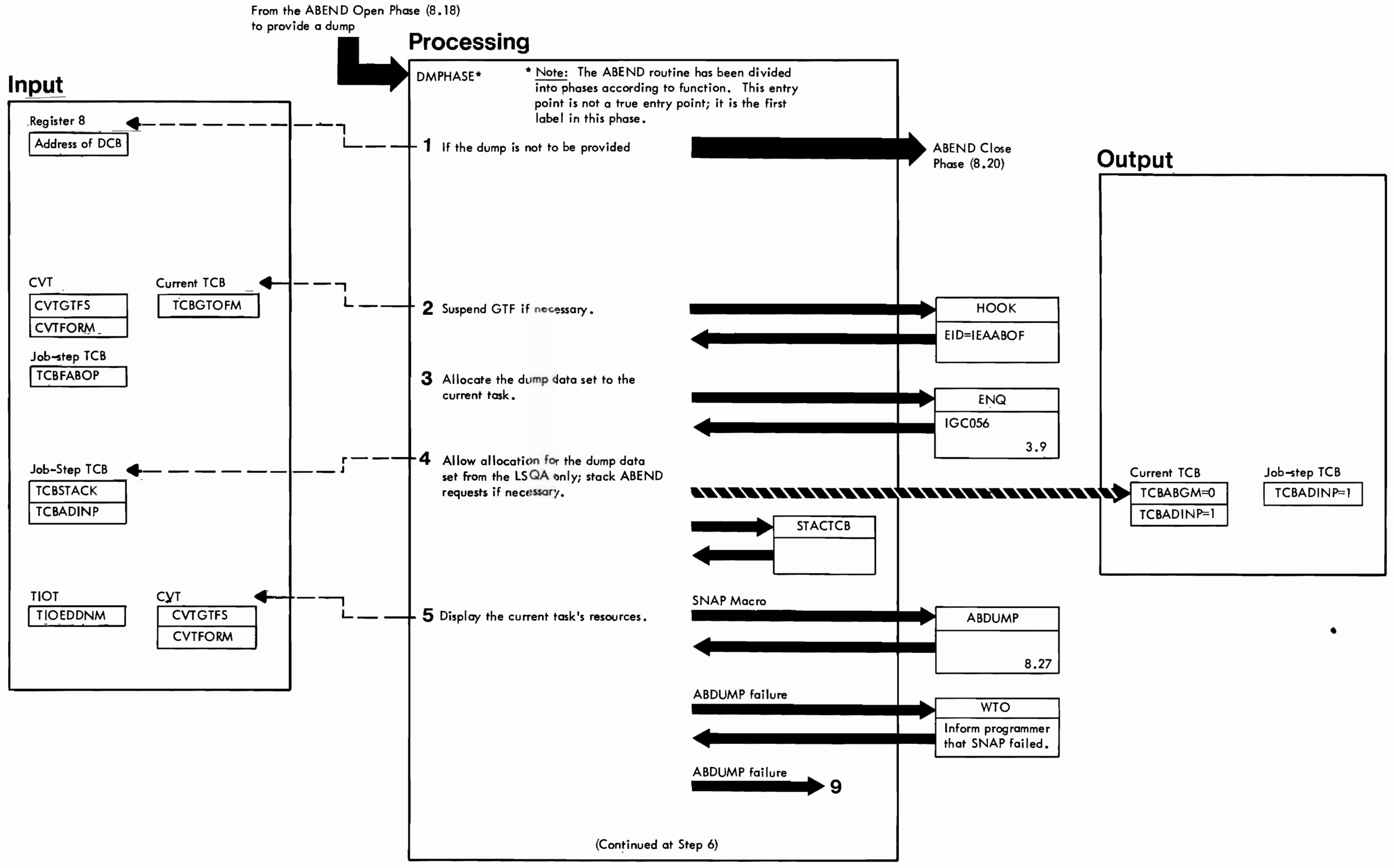

From the ABEND Open Phase (8.18) to provide a dump
Input
Register 8
Address of DCB
CVT
CVTGTFS
CVTFORM
Current TCB
TCBGTOFM
Job-step TCB
TCBFABOP
Job-Step TCB
TCBSTACK
TCBADINP
TIOT
TIOEDDNM
CVT
CVTGTFS
CVTFORM
Processing
DMPHASE*
* Note: The ABEND routine has been divided into phases according to function. This entry point is not a true entry point; it is the first label in this phase.
1 If the dump is not to be provided
ABEND Close Phase (8.20)
2 Suspend GTF if necessary.
HOOK
EID=IEAABOF
3 Allocate the dump data set to the current task.
ENQ
IGC056
3.9
4 Allow allocation for the dump data set from the LSQA only; stack ABEND requests if necessary.
STACTCB
SNAP Macro
5 Display the current task's resources.
ABDUMP
8.27
ABDUMP failure
WTO
Inform programmer that SNAP failed.
ABDUMP failure
9
(Continued at Step 6)
Output
Current TCB
TCBABGM=0
TCBADINP=1
Job-step TCB
TCBADINP=1

Diagram 8.19 (Steps 1-5) ABEND ABDUMP Phase (Module IEAVTM01)

NOTES	ROUTINE NAME	LABEL
2 Under the following circumstances, GTF is suspended to minimize changes to the trace: • GTF is active (CVTGTFS = 1). • The "in-core" trace was specified with the formatting option (CVTFORM-1). • The current task has not already suspended tracing (TCBGTOFM = 1). • The user requested a SYSABEND dump (TCBFABOP indicates SYSABEND).	ABDUMP Phase	
3		BYGTF
4 The current ABEND request is stacked if it is awaiting the completion of some event. Allocation for the dump data set is only possible from the LSQA because the dump data set requires a large amount of storage and the SQA is a critical system resource. The ABDUMP-in-progress flag (TCBADINP) is set prior to invoking the ABDUMP routine.		
5 The ABDUMP parameter list is constructed with the appropriate options specified: • SYSABEND -- entire dump • SYSUDUMP -- problem-program save areas and system control blocks associated with the problem program. • GTF -- only if a SYSABEND DD was specified, and GTF is active.		
Prior to issuing the SNAP macro instruction, interruptions are disabled by MODESET, and the valid recursion flag is set (TCBABCUR=X'12'). This flag is cleared upon completion of ABDUMP processing. If		SNAPCURR
ABDUMP processing fails (return code is greater than 0), a message is constructed informing the programmer that the dump failed. The valid recursion flag is cleared and then set (TCBABCUR = X'14') during WTO processing.		DMPMSG
Upon return from WTO, the valid recursion flag is cleared (TCBABCUR=0), and the message buffer storage acquired earlier is freed (by RMBRANCH). The ABDUMP-in-progress flag (TCBADINP) is turned off in the current and job-step TCBs. Tasks that were stacked are restarted (using RESETHI), asynchronous exits are per-		AFTADWTP
mitted, and registers are restored before entering cleanup processing (Step 9). (See <u>OS/VS2 Job Management Logic</u> for information on the WTO routine.)		BYDMPDSP

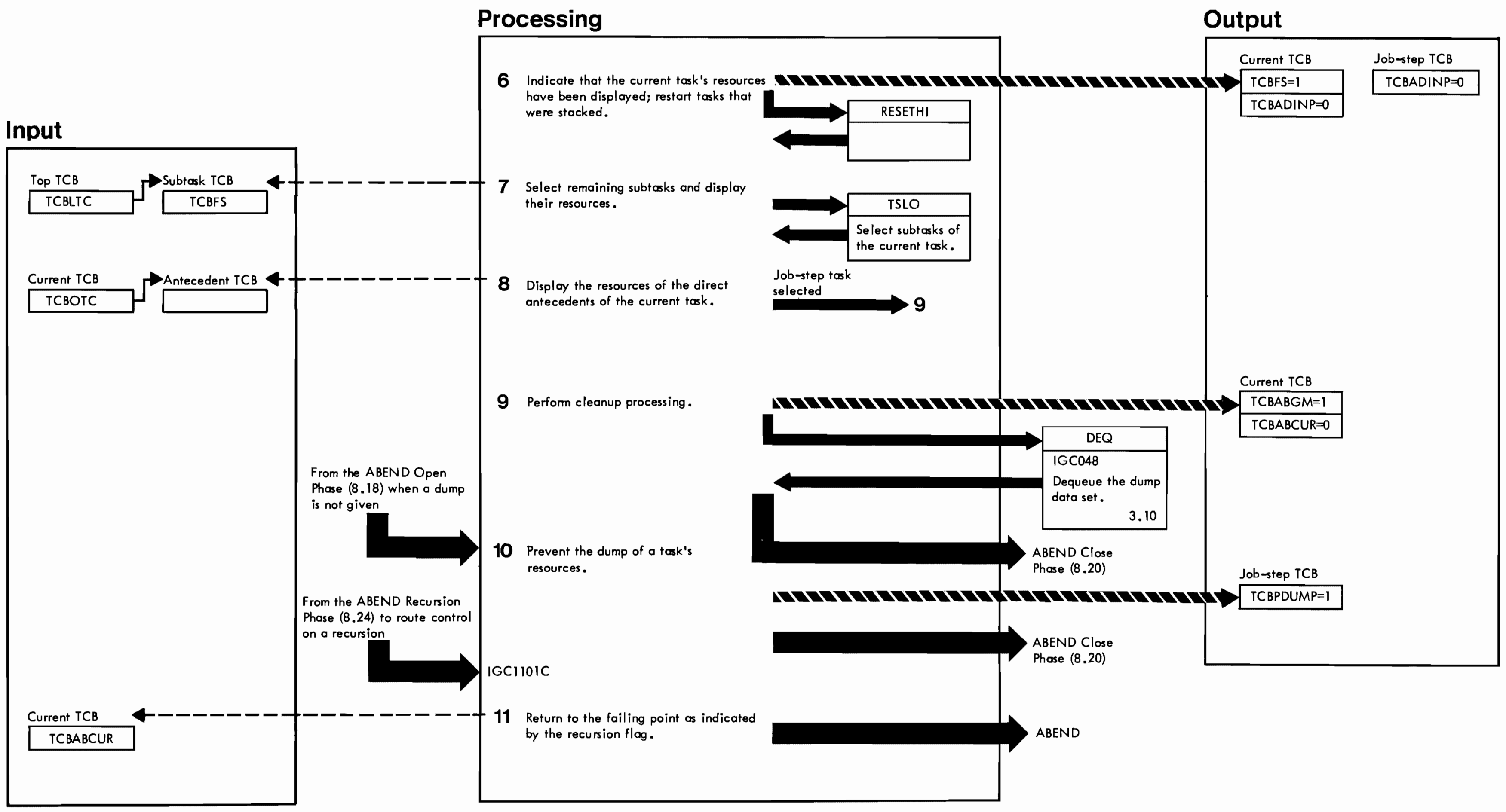

Input
Top TCB
TCBLTC
Subtask TCB
TCBFS
Current TCB
TCBOTC
Antecedent TCB
From the ABEND Open Phase (8.18) when a dump is not given
From the ABEND Recursion Phase (8.24) to route control on a recursion
Current TCB
TCBABCUR
Processing
6 Indicate that the current task's resources have been displayed; restart tasks that were stacked.
RESETHI
7 Select remaining subtasks and display their resources.
TSLO
Select subtasks of the current task.
8 Display the resources of the direct antecedents of the current task.
Job-step task selected
9
9 Perform cleanup processing.
DEQ
IGC048
Dequeue the dump data set.
3.10
10 Prevent the dump of a task's resources.
ABEND Close Phase (8.20)
IGC1101C
ABEND Close Phase (8.20)
11 Return to the failing point as indicated by the recursion flag.
ABEND
Output
Current TCB
TCBFS=1
TCBADINP=0
Job-step TCB
TCBADINP=0
Current TCB
TCBABGM=1
TCBABCUR=0
Job-step TCB
TCBPDUMP=1

Diagram 8.19 (Steps 6-11) ABEND ABDUMP Phase (Module IEAVTM01)

NOTES	ROUTINE NAME	LABEL
6	ABDUMP Phase	ABDPST
7 The TSLO subroutine selects the lowest-level task in the chain of terminating tasks first. On the initial entry into TSLO, the input consists of the complement (negative) of the top terminating task's TCB address. If the top terminating task has a subtask (TCBLTC), the address of the lowest-level subtask's TCB is the output for the first entry; otherwise, the input TCB address is also the output. For subsequent entries into TSLO, the input consists of the address of the last TCB selected; the selection of the lowest-level task proceeds as follows:	TSLO	TSLOLOOP
1. If the task represented by the input TCB address has a same-level task (TCBNTC), the lowest-level subtask of that task is selected. If the same-level task has no subtasks, then the same-level task is selected and constitutes the output for this entry into TSLO.		TSLOSTST TSLOSIS TSLOEND
2. If the task represented by the input TCB address does not have a same-level task, the originating task (TCBOTC) of the input task is selected as output from TSLO.		TSLOMOM
If the subtask has not been dumped previously, the following processing is performed: 1. The current ABEND request is stacked by the STACTCB subroutine, if necessary. 2. The ABDUMP-in-progress flag (TCBADINP) is set in the current and job-step TCBs. 3. Registers are saved in the recursion save area and the valid recursion flag is set (TCBABCUR = X'12'). 4. ABDUMP is invoked by the SNAP macro instruction. A subtask dump is identified by an ID of 001. After ABDUMP processing has been completed, the following functions are performed: 1. The valid recursion flag is cleared (TCBABCUR=0). 2. TCBFS is set to 1, indicating that the subtask has been dumped. 3. The ABDUMP-in-progress flag is turned off. 4. Tasks that were stacked are restarted by the RESETHI subroutine. 5. The next subtask is selected to be dumped. After all subtasks have been dumped, the resources owned by the current task's immediate antecedents are displayed (Step 8). Should ABDUMP processing fail, the message described in Step 5 is constructed and sent to the programmer.	ABDUMP Phase	

NOTES	ROUTINE NAME	LABEL
8 Only the direct antecedents of the current task which are in the same job step are dumped. The preparations for calling ABDUMP, and the processing after ABDUMP has returned control are similar to that described for a subtask (see Step 7). The resources displayed are the same as for a subtask; the ID for an antecedent is 002. However, the TCBFS flag, indicating that a subtask has been dumped, is not set. If ABDUMP fails, processing continues as in Step 5. The antecedents are selected and processed until the job-step task is encountered; cleanup then begins (Step 9).	ABDUMP Phase	MATRIDMP
9 The following cleanup processing is performed: 1. The GETMAIN default flag (TCBABGM) is set to allow allocation from the SQA. 2. A DEQ macro instruction is issued for the dump data set. 3. The valid recursion flag (TCBABCUR) is set to 0.		DMPCLEAN
4. The Close Phase is entered.		CLOPHASE
10		SETPDMP
11 IGC1101C is the recursion entry point for module IEAVTM01. From this point, based on the exact TCBABCUR flag setting, control passes to a point subsequent to the location at which the failure occurred. Recursion Flag — Entry Point X'11' — FIXLLS X'12' — DMPMSG X'13' — BYOPMSG X'14' — AFTADWTP X'15' — RECT1WTP		

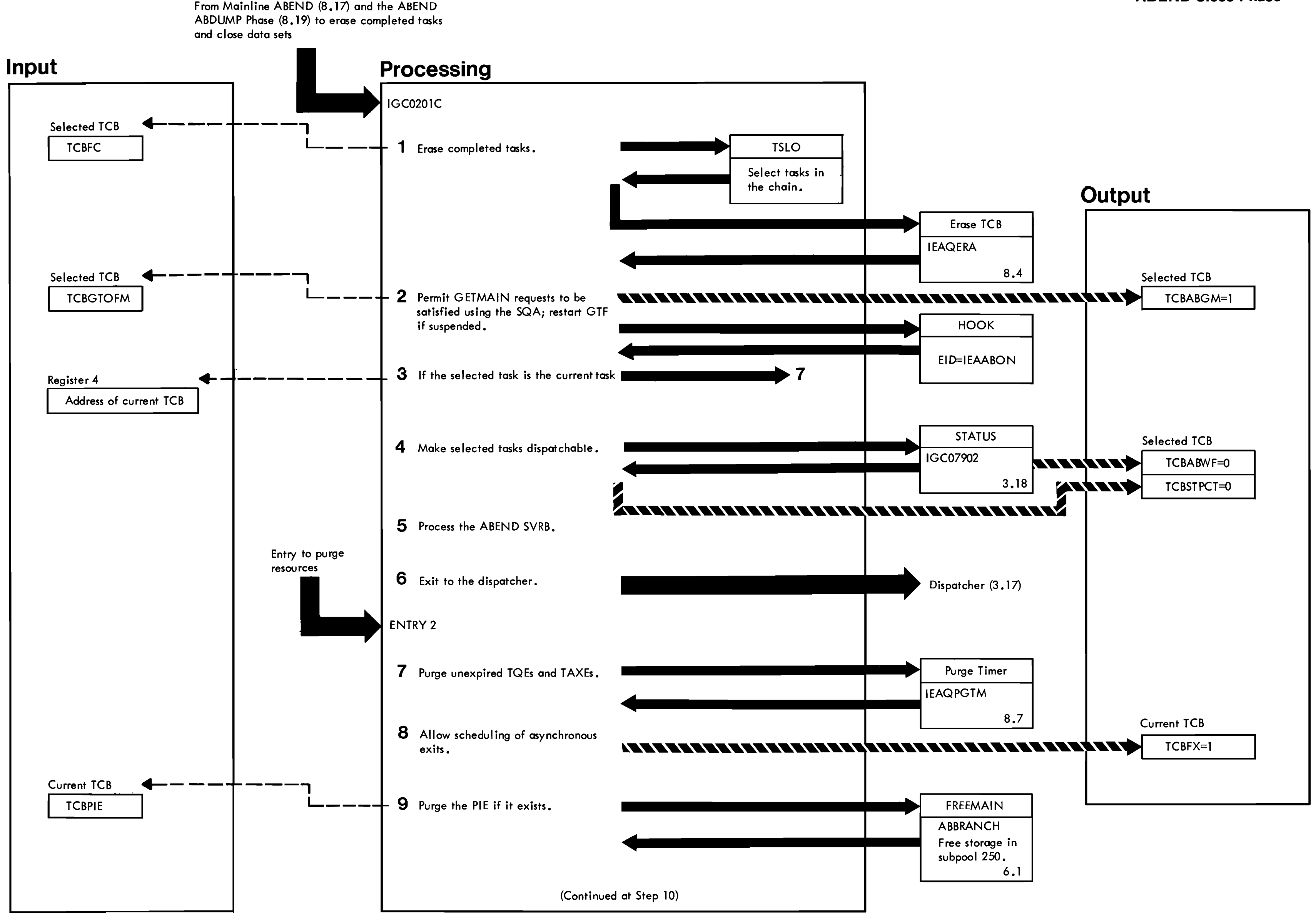

From Mainline ABEND (8.17) and the ABEND ABDUMP Phase (8.19) to erase completed tasks and close data sets
Input
Selected TCB
TCBFC
Selected TCB
TCBGTOFM
Register 4
Address of current TCB
Current TCB
TCBPIE
Processing
IGC0201C
1 Erase completed tasks.
TSLO
Select tasks in the chain.
Erase TCB
IEAQERA
8.4
2 Permit GETMAIN requests to be satisfied using the SQA; restart GTF if suspended.
HOOK
EID=IEAABON
3 If the selected task is the current task
7
4 Make selected tasks dispatchable.
STATUS
IGC07902
3.18
5 Process the ABEND SVRB.
Entry to purge resources
6 Exit to the dispatcher.
Dispatcher (3.17)
ENTRY 2
7 Purge unexpired TQEs and TAXEs.
Purge Timer
IEAQPGTM
8.7
8 Allow scheduling of asynchronous exits.
9 Purge the PIE if it exists.
FREEMAIN
ABBRANCH
Free storage in subpool 250.
6.1
(Continued at Step 10)
Output
Selected TCB
TCBABGM=1
Selected TCB
TCBABWF=0
TCBSTPCT=0
Current TCB
TCBFX=1

Diagram 8.20 (Steps 1-9) ABEND Close Phase (Module IEAVTM02)

NOTES	ROUTINE NAME	LABEL
1 The TSLO subroutine is called to select lower-level tasks in the chain of terminating tasks. For all tasks that have completed execution, the Erase TCB subroutine of the End-of-Task (EOT) routine is called to erase the TCB.	Close Phase	GETLOTCB
2 This processing is performed only for incomplete tasks.		NOTCOMPL
3		GTFON1
4 The selected task is set dispatchable so that closing of each data set can be performed under control of the task that issued the OPEN request.		
5 The ABEND SVRB is moved to the selected task's RB queue. The resume PSW is set to point to entry point ENTRY2 (see Step 7). Registers are moved from the TCB to the ABEND SVRB.		
6 Before exiting to the dispatcher, the STATUS SVC is issued to set the current task nondispatchable. A task switch is performed (by the Task Switch routine) to ensure that the new TCB pointer is set to 0 if it was equal to the old TCB pointer. The dispatcher causes the selected task to receive control at entry point ENTRY2.		DISPAT
7 The EOT Purge Timer subroutine is called to perform the necessary purges. TAXE purges apply only to TSO tasks.		ENTRY2
8 Later close processing may require the use of asynchronous exits.		

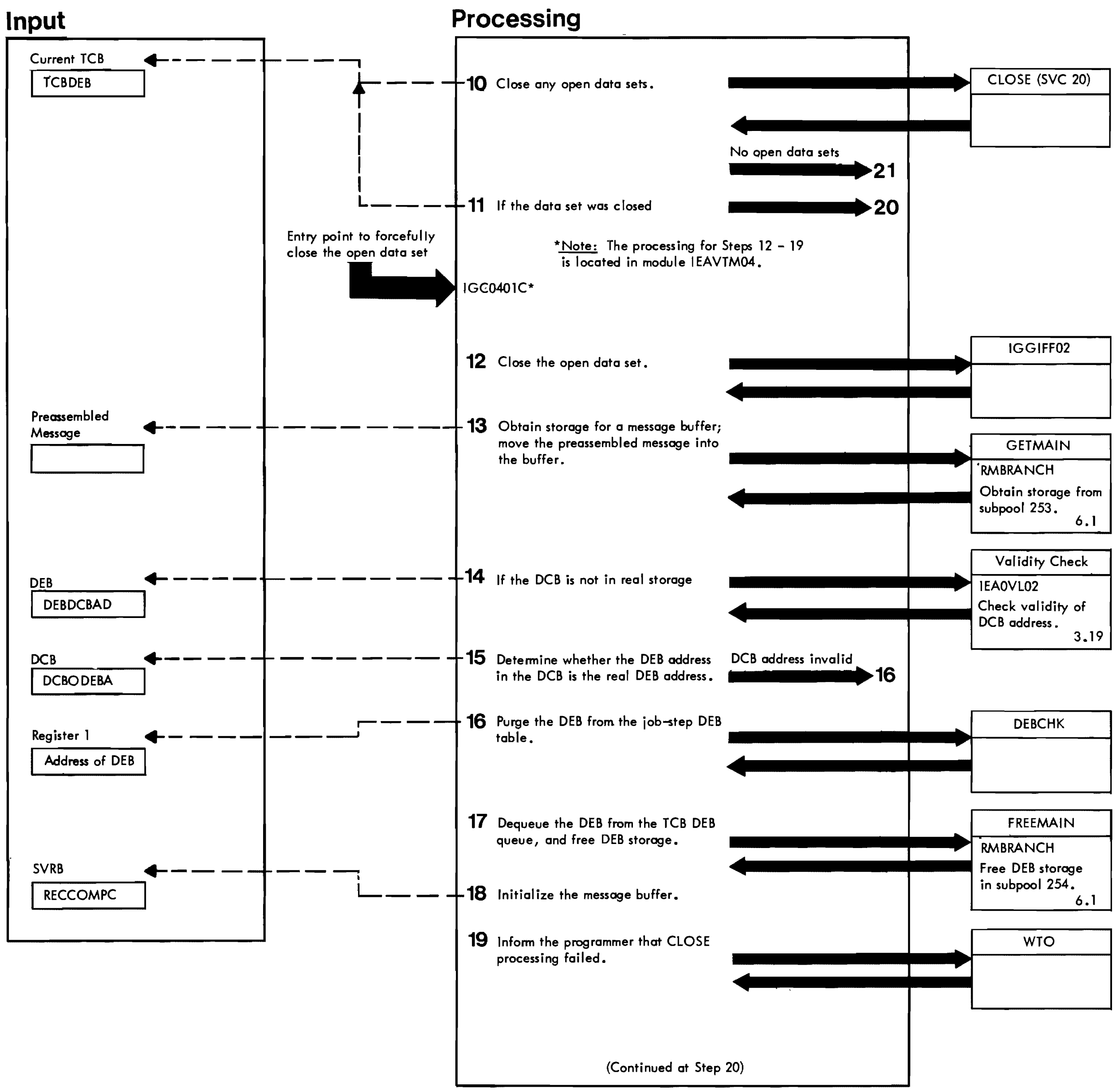

Input
Current TCB
TCBDEB
Preassembled Message
DEB
DEBDCBAD
DCB
DCBODEBA
Register 1
Address of DEB
SVRB
RECCOMPC
Processing
10 Close any open data sets.
No open data sets 21
11 If the data set was closed 20
Entry point to forcefully close the open data set
IGC0401C*
*Note: The processing for Steps 12 - 19 is located in module IEAVTM04.
12 Close the open data set.
13 Obtain storage for a message buffer; move the preassembled message into the buffer.
14 If the DCB is not in real storage
15 Determine whether the DEB address in the DCB is the real DEB address.
DCB address invalid 16
16 Purge the DEB from the job-step DEB table.
17 Dequeue the DEB from the TCB DEB queue, and free DEB storage.
18 Initialize the message buffer.
19 Inform the programmer that CLOSE processing failed.
(Continued at Step 20)
CLOSE (SVC 20)
IGGIFF02
GETMAIN
RMBRANCH
Obtain storage from subpool 253.
6.1
Validity Check
IEAOVL02
Check validity of DCB address.
3.19
DEBCHK
FREEMAIN
RMBRANCH
Free DEB storage in subpool 254.
6.1
WTO

Diagram 8.20 (Steps 10-19) ABEND Close Phase (Modules IEAVTM02 and IEAVTM04)

NOTES	ROUTINE NAME	LABEL
10 The STACTCB subroutine is invoked to a stack ABEND request if required. The OPEN/CLOSE and stacking flags (TCBFOINP and TCBSTACK) should be set so that CLOSE processing is not interrupted. A valid recursion flag (TCBABCUR = X'41') is set during CLOSE processing. (See <u>OS/VS OPEN/CLOSE/EOV Logic</u> for information on the Close routine.)	Close Phase	TESTDEBS
11		SRCHDEBS
12 At this point, IEAVTM04 executes code generated by the IGGIFF02 macro instruction. This code checks for a GAM (graphics access method) DEB, zeros GAM information from UCBs (unit control blocks) associated with GAM DEBs, and issues a FREEMAIN macro instruction to free any TEBs associated with GAM DEBs. For additional information on the IGGIFF02 macro instruction, see <u>OS/VS Graphics Access Method Logic</u>.		FORCLOSE
13		CLOSEMSG
14 Both the ddname and the DEB address in the DCB must be in real storage. If the Validity Check routine finds the DCB address to be valid, the DCB address is referenced to cause a missing page interruption (page fault). (The system is enabled while referencing the DCB.) The DCB is then tested again to determine whether it has been brought into real storage.		DCBLOOP
15 If the DEB address in the DCB is equal to the DEB address, the ddname is obtained from the TIOT and moved into the message buffer. Otherwise, the ddname is obtained from the DCB.		CHKDCB DCBDDNAM
16 See <u>OS/VS OPEN/CLOSE/EOV Logic</u> for information on the DEBCHK routine.		FREEDEB
18 The following elements are moved into the message buffer: • JS or ST (job step or task failure). • DEB address (converted into printable characters). • Completion code from the SVRB (if not 0). The completion code is then zeroed out. • Message length (message text compressed by the Compact subroutine).		MODDNAME
19 Before issuing the WTO instruction, a valid recursion flag (TCBABCUR = X'42') is set and registers are saved in the recursion save area. After the message has been issued, the recursion flag is cleared and the message buffer is freed.		AFTCLWTP

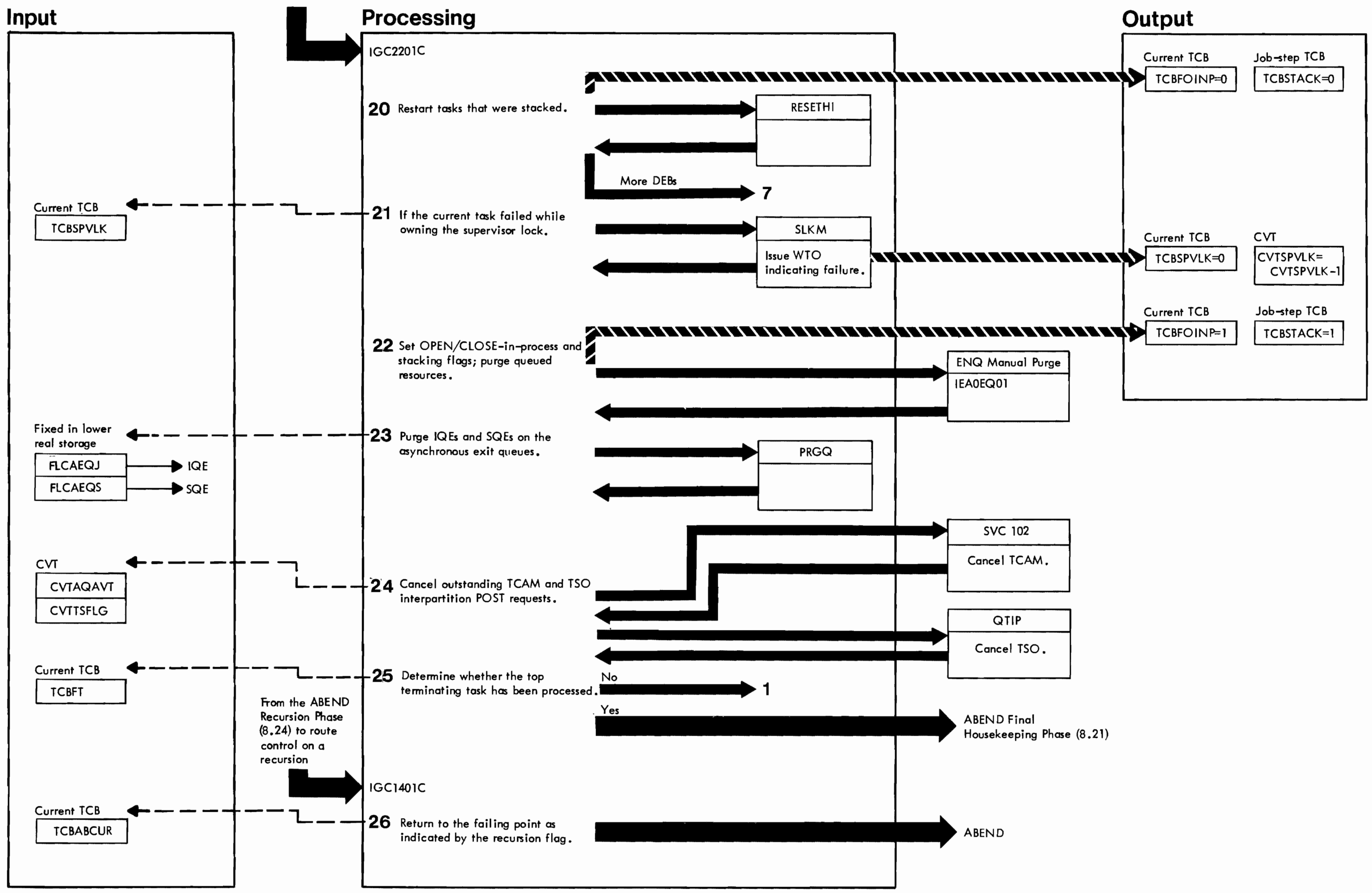
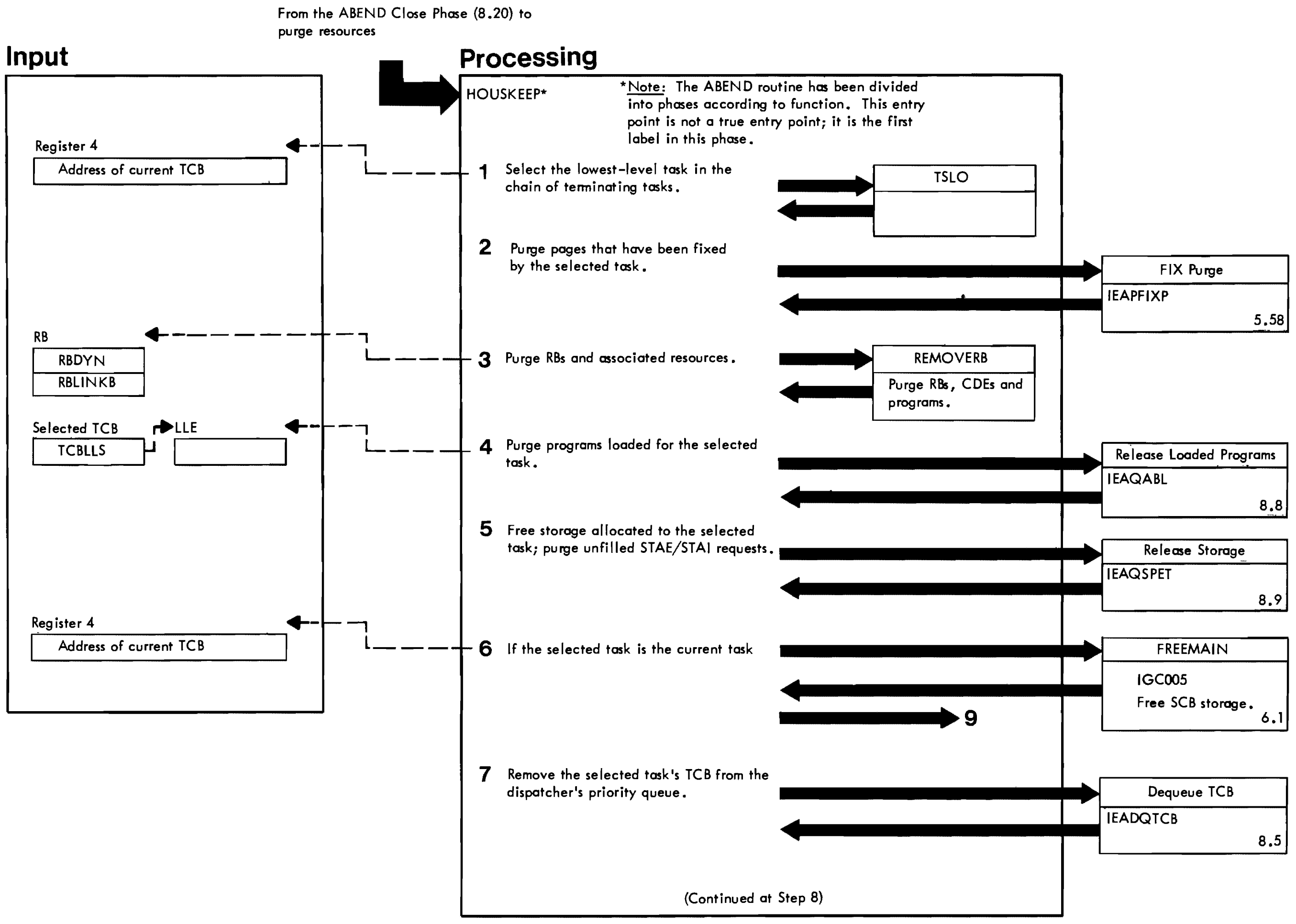

Input
Processing
Output
IGC2201C
20 Restart tasks that were stacked.
RESETHI
More DEBs
7
Current TCB
TCBFOINP=0
Job-step TCB
TCBSTACK=0
Current TCB
TCBSPVLK
21 If the current task failed while owning the supervisor lock.
SLKM
Issue WTO indicating failure.
Current TCB
TCBSPVLK=0
CVT
CVTSPVLK= CVTSPVLK-1
Current TCB
TCBFOINP=1
Job-step TCB
TCBSTACK=1
22 Set OPEN/CLOSE-in-process and stacking flags; purge queued resources.
ENQ Manual Purge
IEA0EQ01
Fixed in lower real storage
FLCAEQJ
FLCAEQS
IQE
SQE
23 Purge IQEs and SQEs on the asynchronous exit queues.
PRGQ
CVT
CVTAQAVT
CVTTSFLG
24 Cancel outstanding TCAM and TSO interpartition POST requests.
SVC 102
Cancel TCAM.
QTIP
Cancel TSO.
Current TCB
TCBFT
25 Determine whether the top terminating task has been processed.
No
1
Yes
ABEND Final Housekeeping Phase (8.21)
From the ABEND Recursion Phase (8.24) to route control on a recursion
IGC1401C
Current TCB
TCBABCUR
26 Return to the failing point as indicated by the recursion flag.
ABEND

Diagram 8.20 (Steps 20-26) ABEND Close Phase (Modules IEAVTM02 and IEAVTM04)

NOTES	ROUTINE NAME	LABEL
20 If the current task does not require suspension, control passes to entry point ENTRY2 to process the next DEB on the TCBDEB queue. This processing (Steps 7-20) is performed until all DEBs on the current task queue have been processed, at which point the supervisor lock flag is tested (Step 21).	Close Phase	CLORESTR
21 A valid recursion flag (TCBABCUR = X'22') is set during SLKM processing, and cleared upon return from SLKM.		SPVLKCHK ENQPURG
22 Because purge processing cannot be interrupted, any current ABEND request stacked. The ENQ Manual Purge routine is called to purge resources. (See "Section 3: Task Supervision" for information on the ENQ Manual Purge routine.) A valid recursion flag (TCBABCUR = X'21') is set to intercept the abnormal termination of EXCP when issued to release a shared DASD device, possibly resulting in an I/O error.		ENQPURGE ENQPG TESTTOP
After ENQ processing has been completed, OPEN/CLOSE, stacking, and valid recursion flags are cleared. The RESETHI subroutine is called to restart tasks.		
23 The PRGQ subroutine first dequeues all IQEs (interruption queue elements) associated with the current task from the AEQJ (IQE asynchronous exit queue). The high-order byte in the last IQE is set to X'FF' to indicate the end of the queue. After all IQEs have been dequeued, all SQEs (supervisor queue elements) associated with the current task are dequeued from the AEQS (SQE asychronous exit queue). The high-order byte in the last SQE is set to X'FF', and control is returned.	PRGQ	PRGQITST PRGQIDQ PRGQSTST PRGQEND
24 If a TCAM task is present (CVTAQAVB) and it is the MCP (Message Control Program), the following processing occurs:	Close Phase	TCAMPOST
1. An SVRB is obtained (using GETMAIN).		
2. The SVRB is queued to the top of the RB queue.		
3. The SVRB is formatted and ABEND's RBABEND=0 save area is copied.		
4. The resume PSW in the second RB is set to label TSOPOST.		
5. Registers are saved in the recursion save area; the valid recursion flag is set (TCBABCUR=X'23').		
6. The input to the TCAM routine is set identical to that produced by the SLIH.		
7. An XCTL instruction is issued to pass control to the TCAM routine IEDQ0T01, which returns control to label TSOPOST if processing is successful.		

NOTES	ROUTINE NAME	LABEL
Should an error occur during IEDQ0T01 processing, or if the MCP task is not present, ABEND tests if TCAM is active (CVTAQAVT). If so, ABEND saves registers in the recursion save area, sets the valid recursion flag (TCBABCUR=X'23'), and issues an SVC 102 instruction to cancel TCAM interpartition Post requests.	Close Phase	TCAMACTV
If TCAM is neither present nor active, the valid recursion flag is cleared (TCBABCUR=0) before ABEND cancels TSO interpartition Post requests.		TSOPOST
See _OS/VS TCAM Logic_ for information on SVC 102. See _OS/VS2 TSO Control Program Logic_ for information on the QTIP routine.		
25 If the current task is the top terminating task, close processing is complete and the Final Housekeeping Phase can be entered. Otherwise, close processing is re-entered to process the lower-level tasks that have not already been processed.		TSONO
26 IGC1401C is the recursion entry point for module IEAVTM04. From this point, based on the exact TCBABCUR flag setting, control passes to a point subsequent to the location at which the failure occurred.		
Recursion Flag _Reentry Point_		
X'41' A search is first made to find the DEB. If found, the macro instruction is executed to close the data set. If the DEB is not found, a message buffer is obtained from subpool 253, and the message is issued to the programmer.		CLOSFAIL NODEB NODDNAME
X'42' AFTCLWTP		

Diagram 8.21 (Steps 1-7)
ABEND Final Housekeeping Phase



*Note: The ABEND routine has been divided into phases according to function. This entry point is not a true entry point; it is the first label in this phase.

1 Select the lowest-level task in the chain of terminating tasks.

2 Purge pages that have been fixed by the selected task.

3 Purge RBs and associated resources.

4 Purge programs loaded for the selected task.

5 Free storage allocated to the selected task; purge unfilled STAE/STAI requests.

6 If the selected task is the current task

7 Remove the selected task's TCB from the dispatcher's priority queue.

(Continued at Step 8)

Diagram 8.21 (Steps 1-7) ABEND Final Housekeeping Phase (Module IEAVTM02)

NOTES	ROUTINE NAME	LABEL
1 The lower-level tasks are purged first to eliminate conflicts arising as a result of shared resources.	Final House-keeping Phase	TCBSELOP
3 The REMOVERB subroutine is called to purge resources associated with the task's RB queue. The input to REMOVERB consists of the address of the RB that is queued to the ABEND SVRB if the selected task is the current task. Otherwise, the top RB on the TCB RB queue is provided as input. REMOVERB performs the following functions: • Makes a serially reusable resource, which is con-trolled by the RB, available for other tasks. • Purges all resource requests not fulfilled at this time. • Purges control blocks and work areas associated with a module that the RB has requested and which has not been loaded. • Frees storage occupied by the RB, unless the RB is static (RBDYN), or is the last RB queued to the current TCB. If the RB is static, the storage cannot be freed because it was not dynamically allocated to the task. If the RB is the last RB for the current task, the storage is needed when the interface to the End-of-Task (EOT) routine is established. (See Chart 8-1 for detailed information on the REMOVERB subroutine.)		RBADDROK
4 The Release Loaded Programs subroutine purges the necessary loaded programs and control blocks.		
5 The Release Storage subroutine frees the storage allocated to the selected task in its own subpool. In addition, storage in the LSQA and SQA is freed from subpool 253. All STA control blocks are freed from subpool 253.		SCBPURGE
6		BYSCB
7 Upon return from the Dequeue TCB subroutine, the TCBFC flag is set to indicate that the task has completed processing.		TCBERASE

Diagram 8.21 (Steps 8-14)
ABEND Final Housekeeping Phase

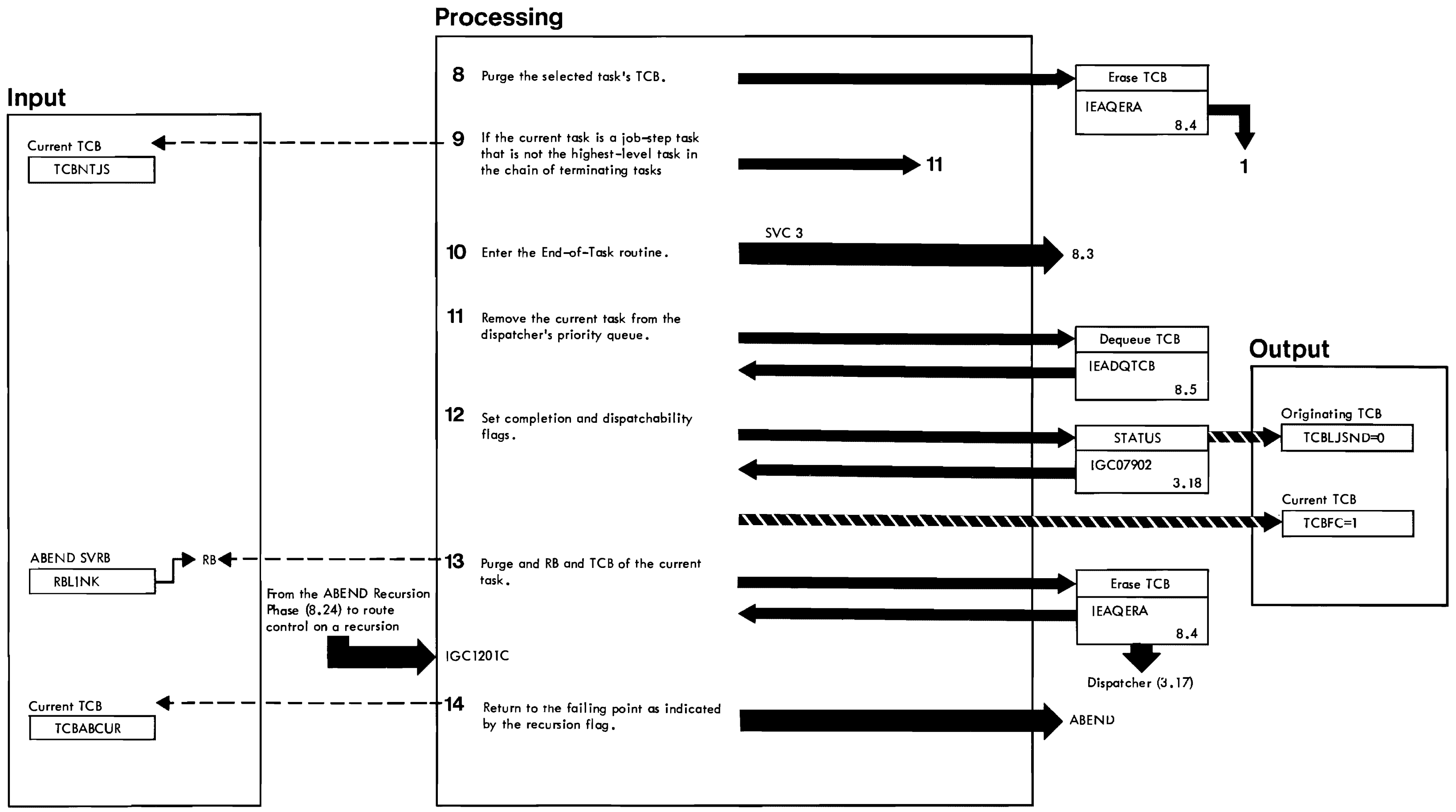

Input
Processing
Output
Current TCB
TCBNTJS
ABEND SVRB
RBLINK
RB
Current TCB
TCBABCUR
8 Purge the selected task's TCB.
9 If the current task is a job-step task that is not the highest-level task in the chain of terminating tasks
11
10 Enter the End-of-Task routine.
SVC 3
8.3
11 Remove the current task from the dispatcher's priority queue.
12 Set completion and dispatchability flags.
13 Purge and RB and TCB of the current task.
From the ABEND Recursion Phase (8.24) to route control on a recursion
IGC1201C
14 Return to the failing point as indicated by the recursion flag.
ABEND
Erase TCB
IEAQERA
8.4
1
Dequeue TCB
IEADQTCB
8.5
STATUS
IGC07902
3.18
Erase TCB
IEAQERA
8.4
Dispatcher (3.17)
Originating TCB
TCBLJSND=0
Current TCB
TCBFC=1

Diagram 8.21 (Steps 8-14) ABEND Final Housekeeping Phase (Module IEAVTM02)

NOTES	ROUTINE NAME	LABEL
8 The Erase TCB subroutine dequeues the selected TCB and frees the storage occupied by the TCB. The next task is then selected and processed (Step 1).	Final House-keeping Phase	EOTERASE
9 If the current task is a job-step task that is not the highest-level task, a test must be made to determine whether multiple job-step failures have occurred.		
10 ABEND places the completion code in register 15 and exits to the End-of-Task routine. EOT posts the originating task's ECB and/or schedules the originating task's ETXR routine. If neither action was specified when the task was created, EOT erases the current task.		
11 The Dequeue TCB subroutine removes the current task.		TCBERASE
12 The nondispatchability flag (TCBLJSND) was set earlier by ABEND when it was determined that a multiple job-step failure had occurred. At that time it was necessary to stack the higher-level job step until the lower-level job step had been terminated. A task switch is performed if necessary.		
13 If the ABEND SVRB points to the TCB, the Erase TCB subroutine purges the last RB and the TCB of the current task. EOT exits to the Dispatcher, which passes control to the originating task. The originating task then reenters ABEND processing at the point at which it was suspended. Otherwise, the RB queued to the ABEND SVRB is dequeued from the selected task's RB queue. If the RB is not an IRB or TIRB, but is a dynamic RB, FREEMAIN is called to free the RB previously queued to the ABEND SVRB. The next RB (RBLINK is then tested.		FINAL
14 IGC1201C is the recursion entry point for module IEAVTM02. From this point, based on the exact TCBABCUR flag setting, control passes to a point subsequent to the location at which the failure occurred. __Recursion Flag__ __Reentry Point__ X'21' ENQPG X'22' ENQPURG X'23' TSOPOST		

Chart 8-1 (page 1 of 2). REMOVERB Subroutine

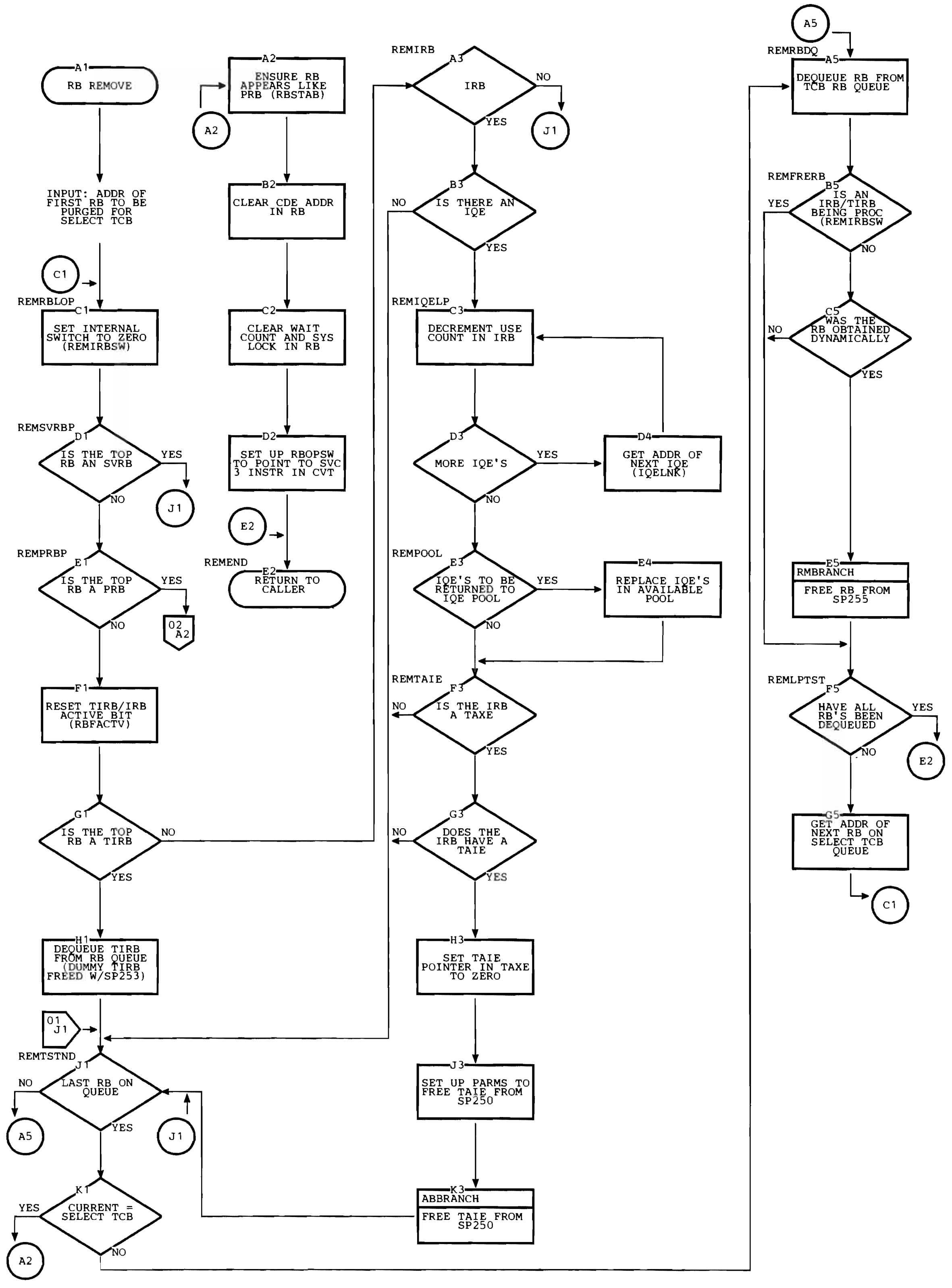
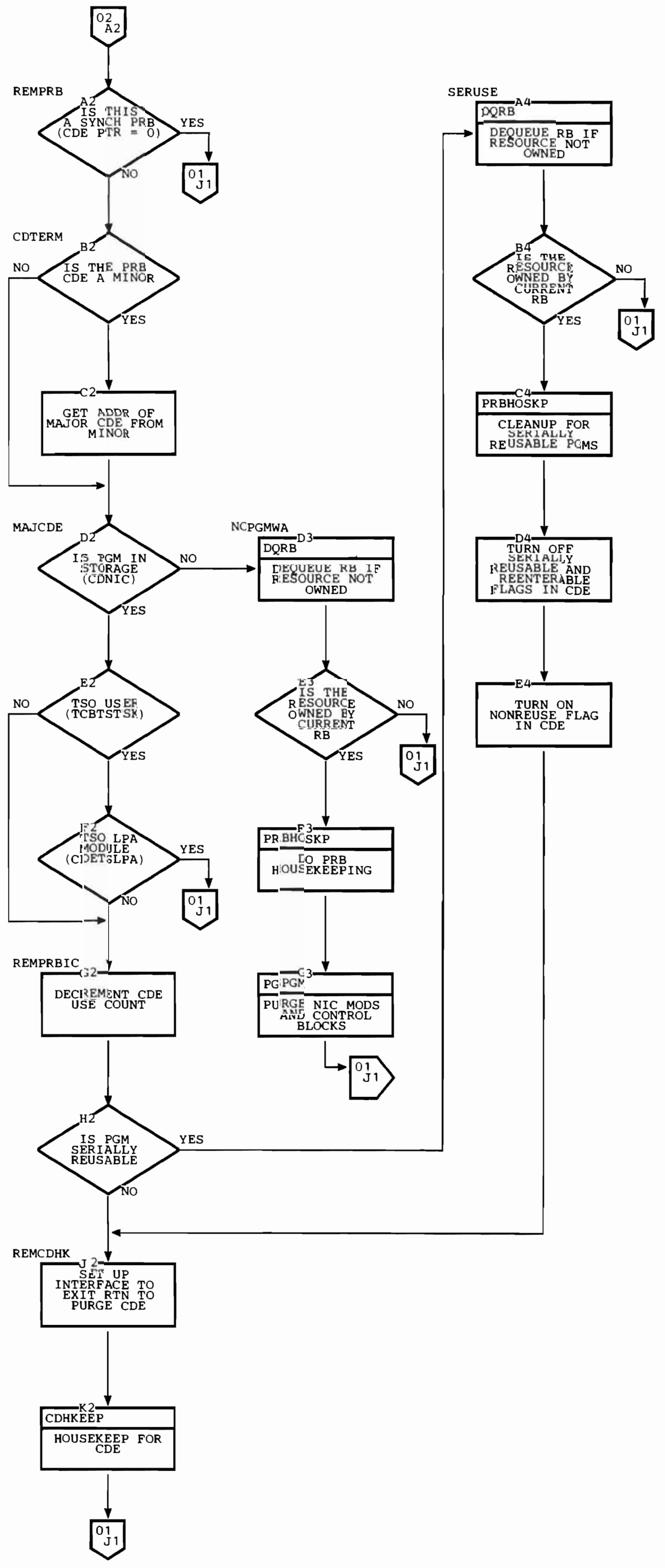
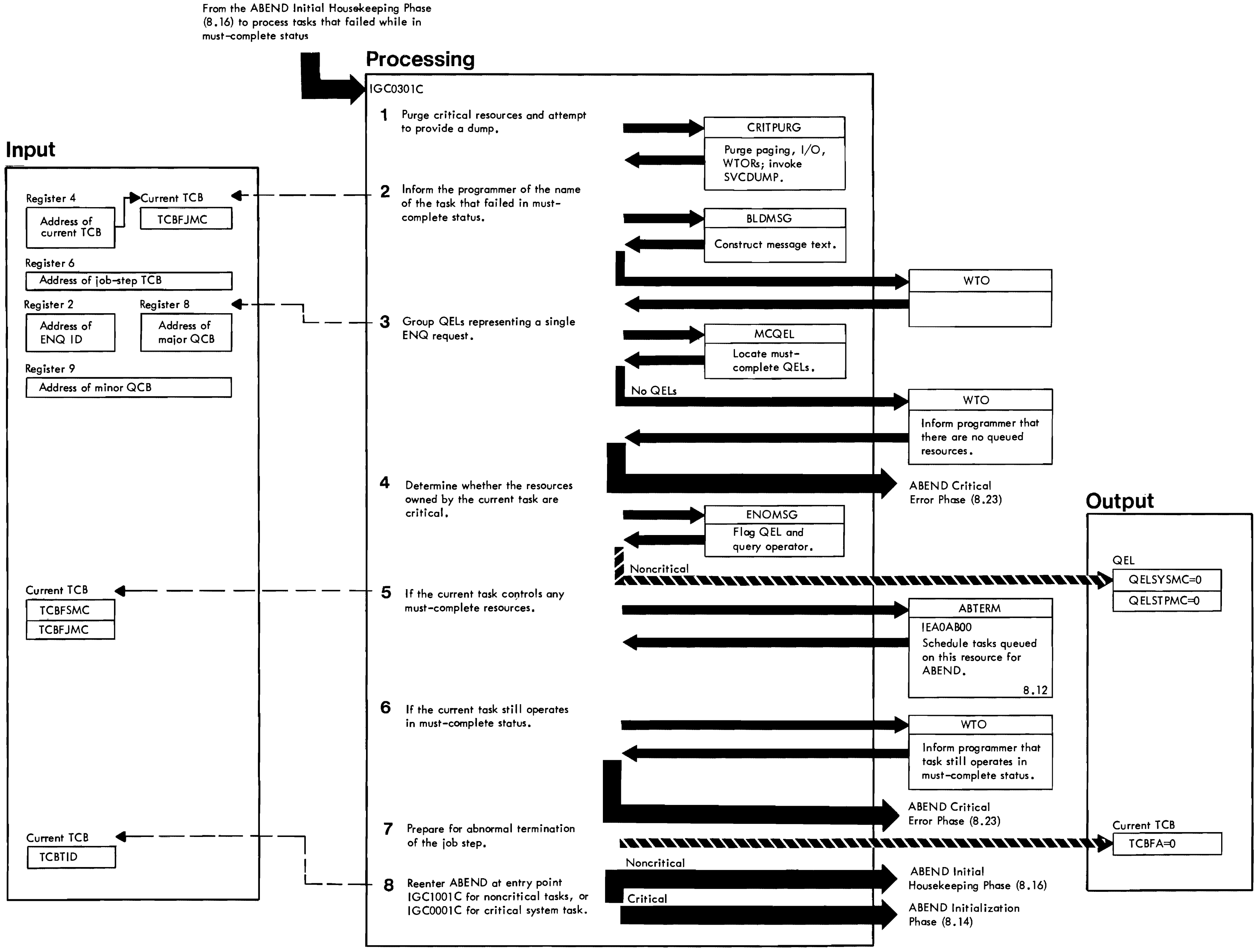

A1 RB REMOVE
INPUT: ADDR OF FIRST RB TO BE PURGED FOR SELECT TCB
C1
REMRBLOP
C1 SET INTERNAL SWITCH TO ZERO (REMIRBSW)
REMSVRBP
D1 IS THE TOP RB AN SVRB
YES
J1
NO
REMPRBP
E1 IS THE TOP RB A PRB
YES
02 A2
NO
F1 RESET TIRB/IRB ACTIVE BIT (RBFACTV)
G1 IS THE TOP RB A TIRB
NO
YES
H1 DEQUEUE TIRB FROM RB QUEUE (DUMMY TIRB FREED W/SP253)
01 J1
REMTSTND
J1 LAST RB ON QUEUE
NO
A5
YES
J1
K1 CURRENT = SELECT TCB
YES
A2
NO
A2
A2 ENSURE RB APPEARS LIKE PRB (RBSTAB)
B2 CLEAR CDE ADDR IN RB
C2 CLEAR WAIT COUNT AND SYS LOCK IN RB
D2 SET UP RBOPSW TO POINT TO SVC 3 INSTR IN CVT
E2
REMEND
E2 RETURN TO CALLER
REMIRB
A3 IRB
NO
J1
YES
B3 IS THERE AN IQE
NO
YES
REMIQELP
C3 DECREMENT USE COUNT IN IRB
D3 MORE IQE'S
YES
D4 GET ADDR OF NEXT IQE (IQELNK)
NO
REMPOOL
E3 IQE'S TO BE RETURNED TO IQE POOL
YES
E4 REPLACE IQE'S IN AVAILABLE POOL
NO
REMTAIE
F3 IS THE IRB A TAXE
NO
YES
G3 DOES THE IRB HAVE A TAIE
NO
YES
H3 SET TAIE POINTER IN TAXE TO ZERO
J3 SET UP PARMS TO FREE TAIE FROM SP250
K3 ABBRANCH
FREE TAIE FROM SP250
A5
REMRBDQ
A5 DEQUEUE RB FROM TCB RB QUEUE
REMFRERB
B5 IS AN IRB/TIRB BEING PROC (REMIRBSW
YES
NO
C5 WAS THE RB OBTAINED DYNAMICALLY
NO
YES
E5 RMBRANCH
FREE RB FROM SP255
REMLPTST
F5 HAVE ALL RB'S BEEN DEQUEUED
YES
E2
NO
G5 GET ADDR OF NEXT RB ON SELECT TCB QUEUE
C1

Chart 8-1 (page 2 of 2). **REMOVERB Subroutine**



02 A2
REMPRB
A2 IS THIS A SYNCH PRB (CDE PTR = 0)
YES
NO
01 J1
CDTERM
B2 IS THE PRB CDE A MINOR
NO
YES
C2 GET ADDR OF MAJOR CDE FROM MINOR
MAJCDE
D2 IS PGM IN STORAGE (CDNIC)
NO
YES
NCPGMWA
D3 DQRB DEQUEUE RB IF RESOURCE NOT OWNED
E2 TSO USER (TCBTSTSK)
NO
YES
E3 IS THE RESOURCE OWNED BY CURRENT RB
NO
YES
01 J1
F2 TSO LPA MODULE (CDETSLPA)
YES
NO
01 J1
F3 PRBHOSKP DO PRB HOUSEKEEPING
REMPRBIC
G2 DECREMENT CDE USE COUNT
G3 PGPGM PURGE NIC MODS AND CONTROL BLOCKS
01 J1
H2 IS PGM SERIALLY REUSABLE
YES
NO
REMCDHK
J2 SET UP INTERFACE TO EXIT RTN TO PURGE CDE
K2 CDHKEEP HOUSEKEEP FOR CDE
01 J1
SERUSE
A4 DQRB DEQUEUE RB IF RESOURCE NOT OWNED
B4 IS THE RESOURCE OWNED BY CURRENT RB
NO
YES
01 J1
C4 PRBHOSKP CLEANUP FOR SERIALLY REUSABLE PGMS
D4 TURN OFF SERIALLY REUSABLE AND REENTERABLE FLAGS IN CDE
E4 TURN ON NONREUSE FLAG IN CDE



From the ABEND Initial Housekeeping Phase (8.16) to process tasks that failed while in must-complete status

Input

Register 4: Address of current TCB → Current TCB: TCBFJMC

Register 6: Address of job-step TCB

Register 2: Address of ENQ ID

Register 8: Address of major QCB

Register 9: Address of minor QCB

Current TCB: TCBFSMC, TCBFJMC

Current TCB: TCBTID

Processing

IGC0301C

1 Purge critical resources and attempt to provide a dump.

CRITPURG — Purge paging, I/O, WTORs; invoke SVCDUMP.

2 Inform the programmer of the name of the task that failed in must-complete status.

BLDMSG — Construct message text.

WTO

3 Group QELs representing a single ENQ request.

MCQEL — Locate must-complete QELs.

No QELs → WTO — Inform programmer that there are no queued resources.

→ ABEND Critical Error Phase (8.23)

4 Determine whether the resources owned by the current task are critical.

ENOMSG — Flag QEL and query operator.

Noncritical → QEL: QELSYSMC=0, QELSTPMC=0

5 If the current task controls any must-complete resources.

ABTERM — IEA0AB00 Schedule tasks queued on this resource for ABEND. 8.12

6 If the current task still operates in must-complete status.

WTO — Inform programmer that task still operates in must-complete status.

→ ABEND Critical Error Phase (8.23)

7 Prepare for abnormal termination of the job step.

→ Current TCB: TCBFA=0

8 Reenter ABEND at entry point IGC1001C for noncritical tasks, or IGC0001C for critical system task.

Noncritical → ABEND Initial Housekeeping Phase (8.16)

Critical → ABEND Initialization Phase (8.14)

Output

QEL: QELSYSMC=0, QELSTPMC=0

Current TCB: TCBFA=0

Diagram 8.22 ABEND Must-Complete Phase (Module IEAVTM03)

NOTES	ROUTINE NAME	LABEL
1 The CRITPURG subroutine calls the following routines to purge resources for the current task: 1. Termination Interface (Diagram 5.57) -- purge paging activity. 2. Purge I/O -- purge I/O activity. 3. WTOR Purge -- purge WTOR requests.		
Before calling SVCDUMP to take a system dump, a valid recursion flag (TCBABCUR = X'34') is set. If the SVCDUMP routine is unsuccessful (return code ≠ 0), a valid recursion flag is again set (TCBABCUR = X'35'), and a message buffer is allocated from subpool 253. After the message text has been built by the internal subroutine BLDMSG, a WTO instruction is issued indicating that the SVCDUMP routine failed to provide a dump. The buffer storage is then freed, recursion flags are cleared, and execution continues.	CRITPURG	CRITDUMP CRITMSG CREAFWTO CRITEND
2 The message buffer is obtained from subpool 253 by the RMBRANCH routine; the message text is built by the BLDMSG subroutine. A valid recursion flag (TCBABCUR = X'31') is set across WTO processing; it is cleared upon completion of processing.	Must-Complete Phase	BYENQOPM
3 Because the current task may have issued more than one ENQ request, it is necessary to isolate all resources allocated as a result of each separate request. However, since there can be only a single resource allocated on a RESERVE request, it is not necessary to try to group RESERVE requests. The MCQEL subroutine is called to locate must-complete QELs for the current task. After a must-complete QEL is located, the ENQ request ID is saved to identify the request. The MCQEL subroutine is invoked again to find additional must-complete QELs that have the same ENQ ID and for which a STATUS SVC instruction has been issued. The STATUS SVC instruction is issued to establish the must-complete environment; the QELSTAT flag is set in one of the QELs to indicate that all resources requested have been allocated and the must-complete environment has been established as a result of that request. If a message is issued to the programmer, a valid recursion flag (TCBABCUR = X'33') is set during WTO processing; it is cleard upon completion of processing. (See Chart 8.2 for detailed information on the MCQEL subroutine.)		ISOLAQEL SAVEID TESTSVC ERRMSG AFERRMSG
4 For each must-complete resource owned by the current task, the ENQMSG subroutine is called to flag the QEL as processed (QELABEND), and determine whether the resource is critical (using WTOR). The ENQMSG subroutine sets the valid recursion flag (TCBABCUR = X'36') for WTOR processing, and clears it upon completion of WTOR processing. If the resource is considered noncritical, the nondispatchability flags are turned off so that a later entry into the ENQ Manual Purge routine causes the resource to be released. A critical resource is flagged as permanently unavailable, and any tasks with outstanding requests for that resource are scheduled for abnormal termination using ABTERM. Any tasks requesting unconditional use of the resource are abnormally terminated by the ENQ Manual Purge routine. The must-complete environment can be eliminated only if all resources associated with the request that established the environment are released. Regardless of whether the resource is critical, the MCQEL subroutine is invoked to select the next must-complete QEL with the same ENQ ID. If a step resource is queued with the SMC = STEP option, the resource is automatically released and the associated job step is abnormally terminated. A STEP resource does not represent a threat to system integrity. A message is issued to the programmer to inform him that the resource was released. A valid recursion flag (TCBABCUR = X'32') is set during WTO processing and cleared upon return.	ENQMSG Must-Complete Phase	ISSUEMSG ENQMCRIT RESETMC TSTRESRV TSTSTEP AFTMCWTP
5 After all must-complete resources have been defined as critical or noncritical, the current task is examined to determine whether it still controls any must-complete resources. If yes, any other tasks (except TSO "out-of-core" users) queued to use the must-complete resource are acheduled for abnormal termination. The resource is flagged as unavailable and the Critical Error Phase is entered to set the current task's job step permanently nondispatchable.		TESTDONE TERMTCB
6 The current task could still be operating in a must-complete state if the task issued its own STATUS SVC instruction.		
8		MCSYSXIT

Chart 8-2. MCQEL Subroutine

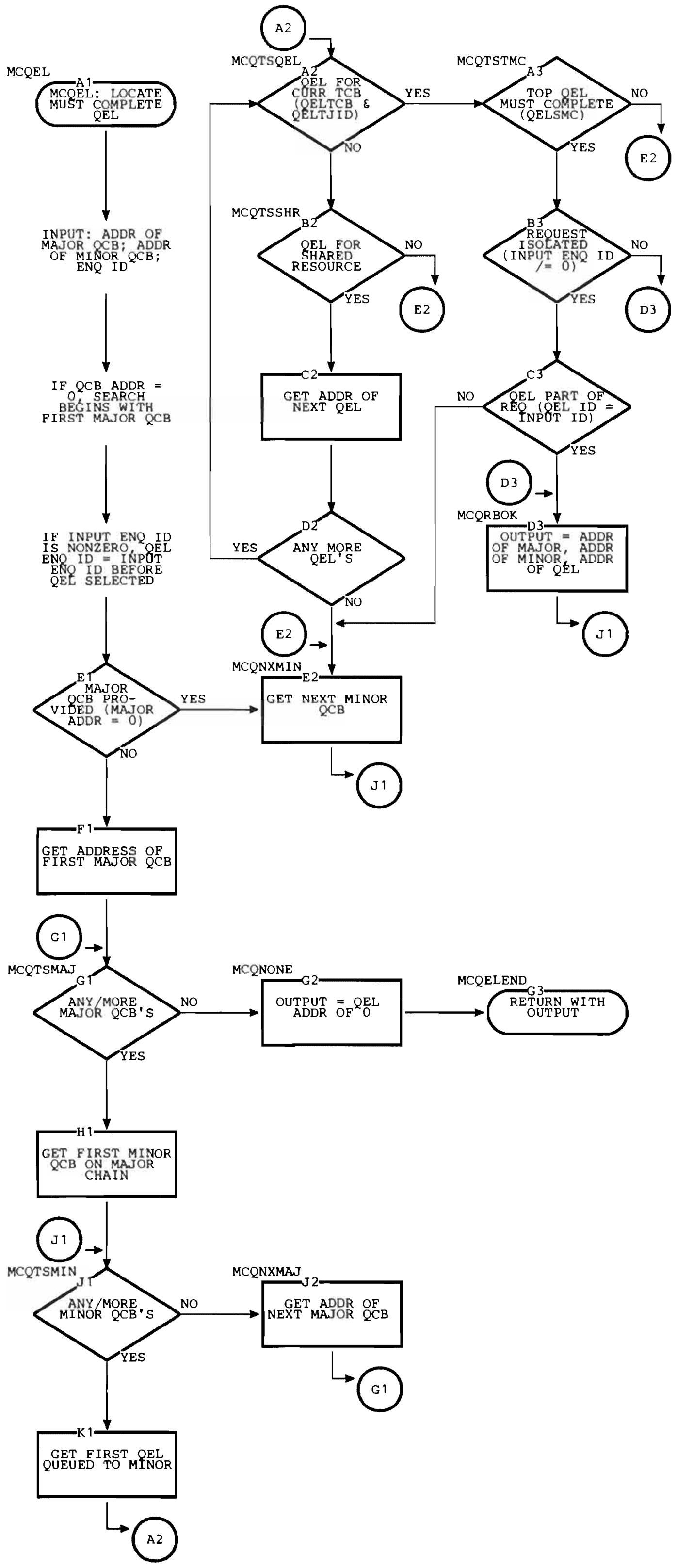
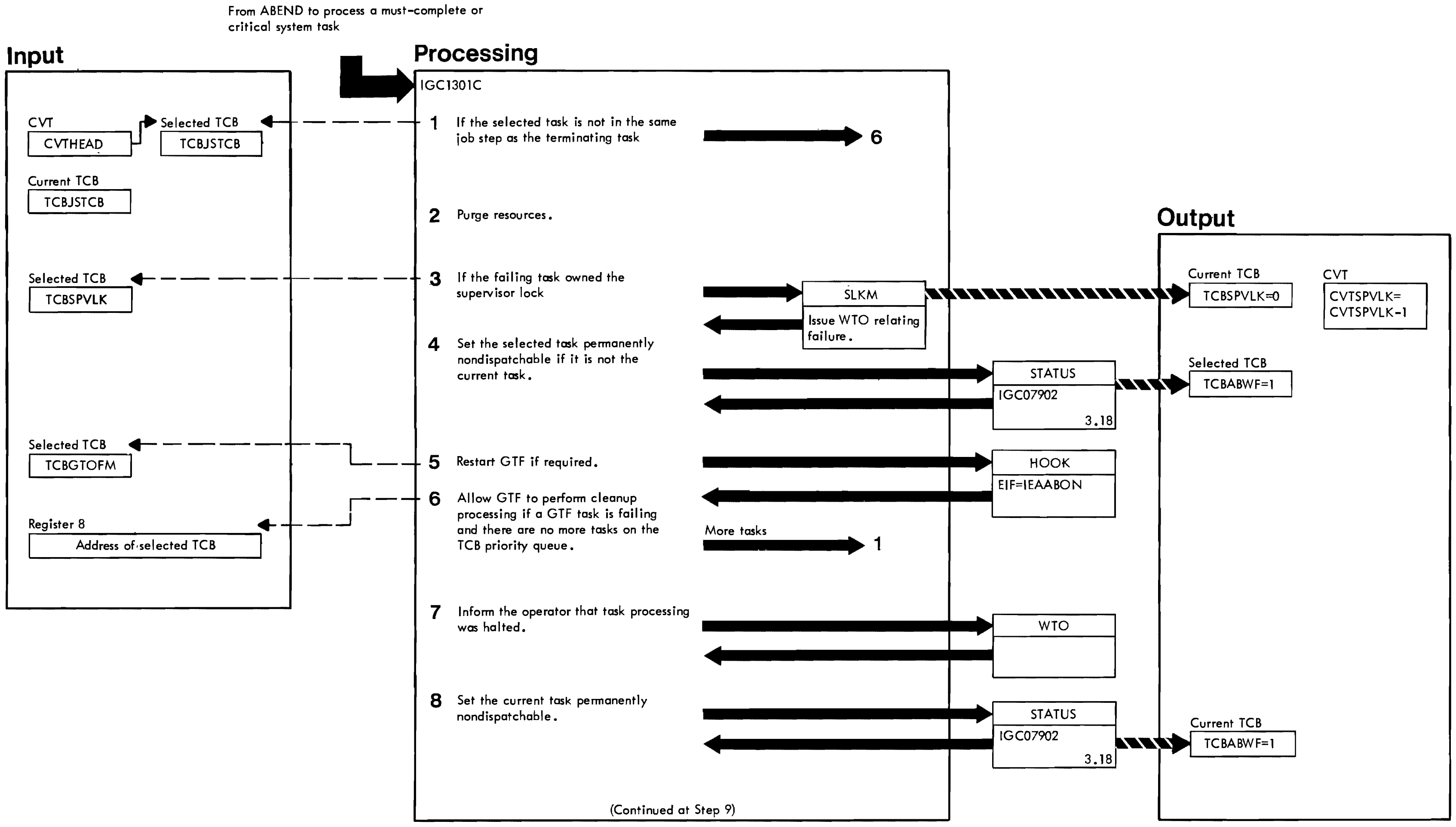

MCQEL
A1 MCQEL: LOCATE MUST COMPLETE QEL
INPUT: ADDR OF MAJOR QCB; ADDR OF MINOR QCB; ENQ ID
IF QCB ADDR = 0, SEARCH BEGINS WITH FIRST MAJOR QCB
IF INPUT ENQ ID IS NONZERO, QEL ENQ ID = INPUT ENQ ID BEFORE QEL SELECTED
E1 MAJOR QCB PROVIDED (MAJOR ADDR = 0)
YES
NO
F1 GET ADDRESS OF FIRST MAJOR QCB
G1
MCQTSMAJ
G1 ANY/MORE MAJOR QCB'S
NO
YES
H1 GET FIRST MINOR QCB ON MAJOR CHAIN
J1
MCQTSMIN
J1 ANY/MORE MINOR QCB'S
NO
YES
K1 GET FIRST QEL QUEUED TO MINOR
A2
A2
MCQTSQEL
A2 QEL FOR CURR TCB (QELTCB & QELTJID)
YES
NO
MCQTSSHR
B2 QEL FOR SHARED RESOURCE
NO
E2
YES
C2 GET ADDR OF NEXT QEL
D2 ANY MORE QEL'S
YES
NO
E2
MCQNXMIN
E2 GET NEXT MINOR QCB
J1
MCQNONE
G2 OUTPUT = QEL ADDR OF 0
MCQNXMAJ
J2 GET ADDR OF NEXT MAJOR QCB
G1
MCQTSTMC
A3 TOP QEL MUST COMPLETE (QELSMC)
NO
E2
YES
B3 REQUEST ISOLATED (INPUT ENQ ID /= 0)
NO
D3
YES
C3 QEL PART OF REQ (QEL ID = INPUT ID)
NO
YES
D3
MCQRBOK
D3 OUTPUT = ADDR OF MAJOR, ADDR OF MINOR, ADDR OF QEL
J1
MCQELEND
G3 RETURN WITH OUTPUT

Diagram 8.23 (Steps 1-8)
ABEND Critical Error Phase


From ABEND to process a must-complete or critical system task

IGC1301C

1 If the selected task is not in the same job step as the terminating task → 6

2 Purge resources.

3 If the failing task owned the supervisor lock

4 Set the selected task permanently nondispatchable if it is not the current task.

5 Restart GTF if required.

6 Allow GTF to perform cleanup processing if a GTF task is failing and there are no more tasks on the TCB priority queue.

More tasks → 1

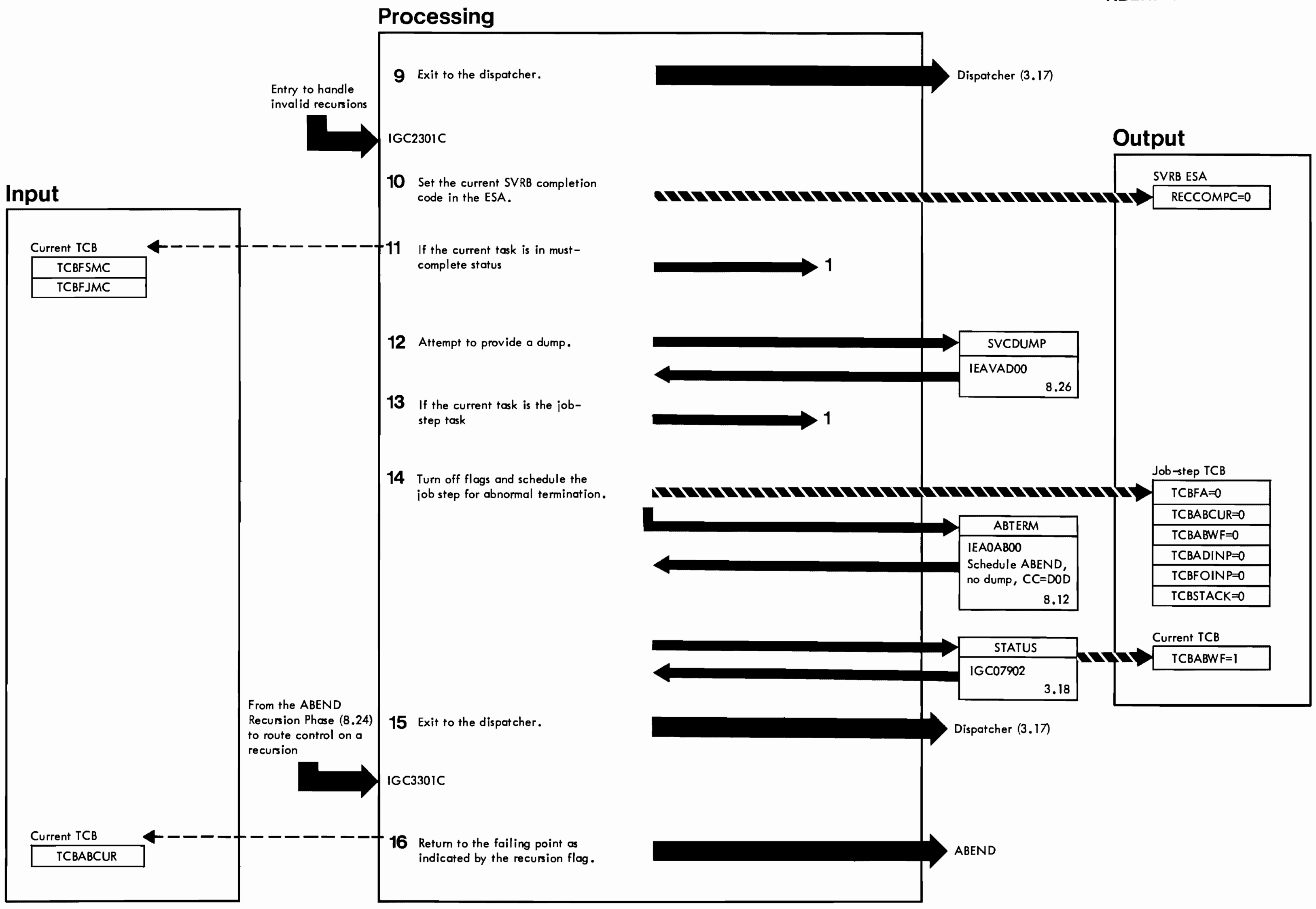
7 Inform the operator that task processing was halted.

8 Set the current task permanently nondispatchable.

(Continued at Step 9)

Diagram 8.23 (Steps 1-8) ABEND Critical Error Phase (Module IEAVTM03)

NOTES	ROUTINE NAME	LABEL
1 All tasks on the task priority queue are checked to determine whether they are in the same job step as the terminating task.	Critical Error Phase	QUIJSPTR
2 The following resources are purged for each task selected:		
1. Entries in the type-1 SVC message table (MSPURG subroutine).		
2. Paging activity (Termination Interface routine).		QUIPAGE
3. Outstanding I/O activity (Purge I/O routine).		QUIPGIO
4. Asynchronous exit queues (PRGQ subroutine).		QUIPRGQ
5. Outstanding WTOR requests (WTOR Purge routine).		QUIWTOR
6. Queued resources (ENQ/DEQ routine).		QUIENQPG
The valid recursion flag (TCBABCUR = X'E1') is set during the processing of the last five routines. Should the processing of any routine result in a recursion, ABEND is reentered at the following points:		
<u>Routine failing</u> — <u>Reentry point</u>		
Termination Interface — QUIPGIO		
Purge I/O — QUIPRGQ		
PRGQ subroutine — QUIWTOR		
WTOR Purge — QUIENQPG		
ENQ/DEQ — QUISLKM		
3 A valid recursion flag (TCBABCUR = X'E1') is set during WTO processing. If a failure occurs, ABEND is reentered at QUITSCUR (Step 4).		QUISLKM
4		QUITSCUR
5 A valid recursion flag (TCBABCUR = X'E1') is set during HOOK processing. If a failure occurs, ABEND is re-entered at QUINXTCB (Step 6).		QUIGTFON
6 A valid recursion flag (TCBABCUR = X'E1') is set during HOOK processing. If a failure occurs, ABEND is reenter-ed at QUIMSG (Step 7).		QUINXTCB
7 The storage for the message buffer is allocated from subpool 253. Valid recursion flags (TCBABCUR = X'E1') are set during GETMAIN/FREEMAIN and WTO processing.		QUIMSG
After issuing the message, the buffer is freed. If a failure occurs, ABEND is reentered at QUIMSGF.		QUIMSGF



9 Exit to the dispatcher.

10 Set the current SVRB completion code in the ESA.

11 If the current task is in must-complete status

12 Attempt to provide a dump.

13 If the current task is the job-step task

14 Turn off flags and schedule the job step for abnormal termination.

15 Exit to the dispatcher.

16 Return to the failing point as indicated by the recursion flag.

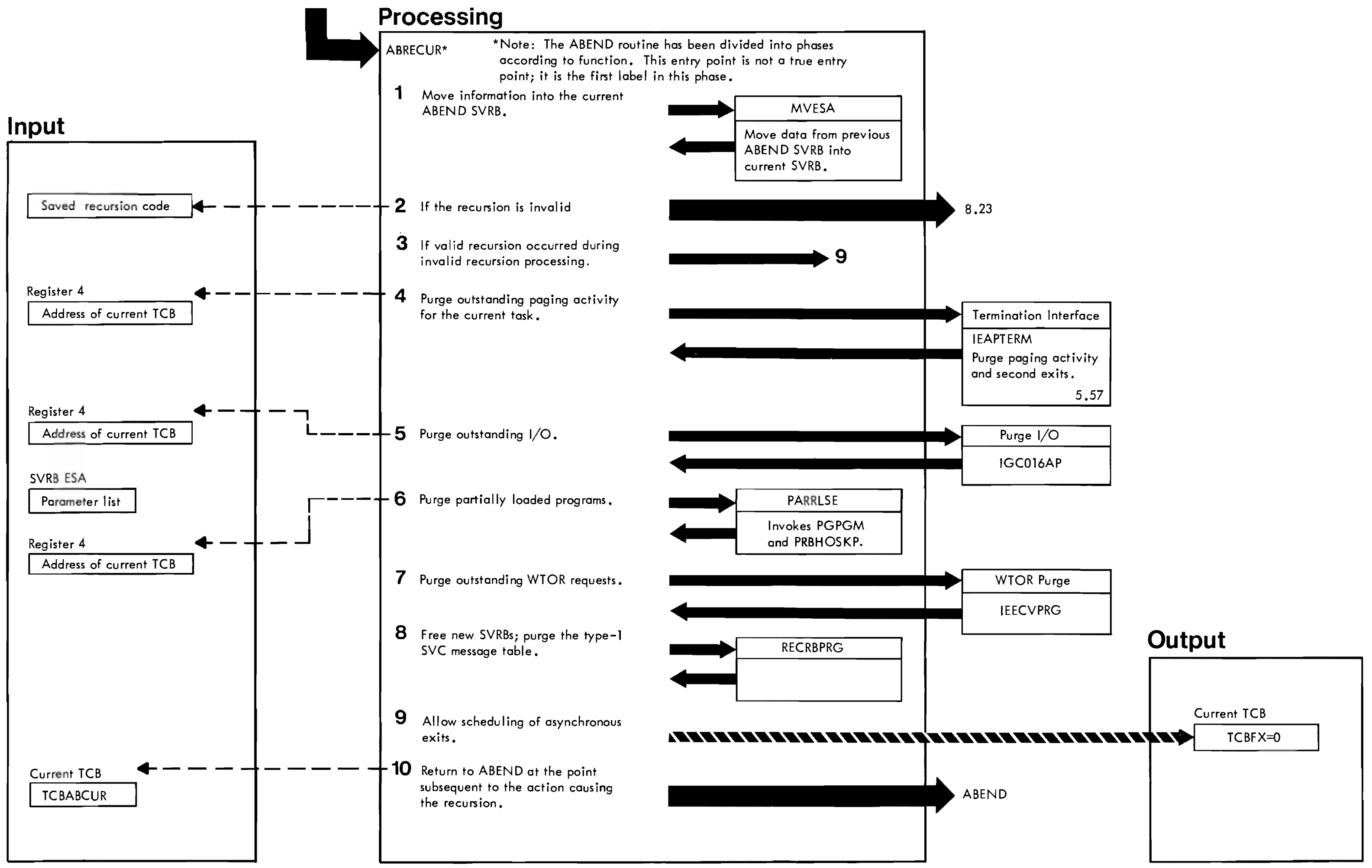
Diagram 8.23 (Steps 9-16) ABEND Critical Error Phase (Module IEAVTM03)

NOTES	ROUTINE NAME	LABEL
9 If a task switch is necessary, it is performed by the Task Switch routine. Control then passes to the dispatcher to dispatch a new task.	Critical Error Phase	QUIESOUT
10 The SVRB cannot be purged because other routines might need to refer to it.		
11 Processing for the job step associated with the current task is halted if the task is in most-complete status. It is not necessary to provide a dump because it was attempted in the Must-Complete Phase.		
12 A valid recursion flag (TCBABCUR = X'E1') is set during SVCDUMP processing. If the dump failed, a message is constructed and sent to the programmer indicating the failure. A valid recursion flag (TCBABCUR=X'E1') is set during WTO processing. The message buffer is freed upon issuing the WTO. If SVCDUMP processing results in a recursion, ABEND is reentered at INVAFDMP. If WTO processing fails, ABEND is reentered at INVAFWTO.		INVDUMP INVAFDMP INVAFWTO
13		INVTSJST
15 Before exiting to the dispatcher, a task switch is ensured. The dispatcher then effects the abnormal termination of the job step.		INVDISP
16 IGC3301C is the recursion entry point for module IEAVTM03. From this point, based on the exact TCBABCUR flag setting, control passes to a point subsequent to the location at which the failure occurred.		

Recursion Flag	Reentry Point
X'31'	BYENQOPM
X'32'	AFTMCWTP
X'33'	AFERRMSG
X'34'	CRITMSG
X'35'	CREAFWTO
X'36'	ENQMCRIT

• Diagram 8.24 ABEND Recursion Phase

From the Initial Housekeeping Phase (8.16) to process the recursion and reenter ABEND if possible



ABRECUR*

*Note: The ABEND routine has been divided into phases according to function. This entry point is not a true entry point; it is the first label in this phase.

1 Move information into the current ABEND SVRB.

2 If the recursion is invalid

3 If valid recursion occurred during invalid recursion processing.

4 Purge outstanding paging activity for the current task.

5 Purge outstanding I/O.

6 Purge partially loaded programs.

7 Purge outstanding WTOR requests.

8 Free new SVRBs; purge the type-1 SVC message table.

9 Allow scheduling of asynchronous exits.

10 Return to ABEND at the point subsequent to the action causing the recursion.

• Diagram 8.24 ABEND Recursion Phase (Module IEAVTM00)

NOTES	ROUTINE NAME	LABEL
2 If an error condition occurs during ABEND processing which leads to a new request for abnormal termination of the task already being terminated, ABEND is re-entered. This reentry (recursion) is scheduled so that ABEND can attempt to complete termination procedures. If ABEND anticipates this reentry, and has steps to deal with it, the recursion is considered valid (see Table of Valid ABEND Recursions). If the recursion is unexpected, it is considered invalid (the saved recursion code = 0); the Critical Error Phase is entered.	Recur-sion Phase	INVRECUR
3		VALIDREC
8 No RBs are purged until the eleventh or subsequent recursion, then all SVRBs existing between the current ABEND SVRB and the previous ABEND SVRB are dequeued and their storage freed. This is done to minimize the amount of queue space required by ABEND.		
9 Asynchronous exits were allowed prior to the recursion and prevented because of the recursion. The TCBABCUR field is reset with the saved recursion code.		NOPURGES
10 Depending on the TCBABCUR flag setting, ABEND is initially reentered at the following recursion entry points:		

TCBABCUR	Module	Recursion Entry Point
X'01'	IEAVTM00	MCTEST
X'11' - X'15'	IEAVTM01	IGC1101C
X'21' - X'25'	IEAVTM02	IGC1201C
X'31' - X'36', X'E1'	IEAVTM03	IGC3301C
X'41' - X'42'	IEAVTM04	IGC1401C

Based on the exact TCBABCUR flag setting, control passes from this recursion entry point to a point subsequent to the location at which the failure occurred.

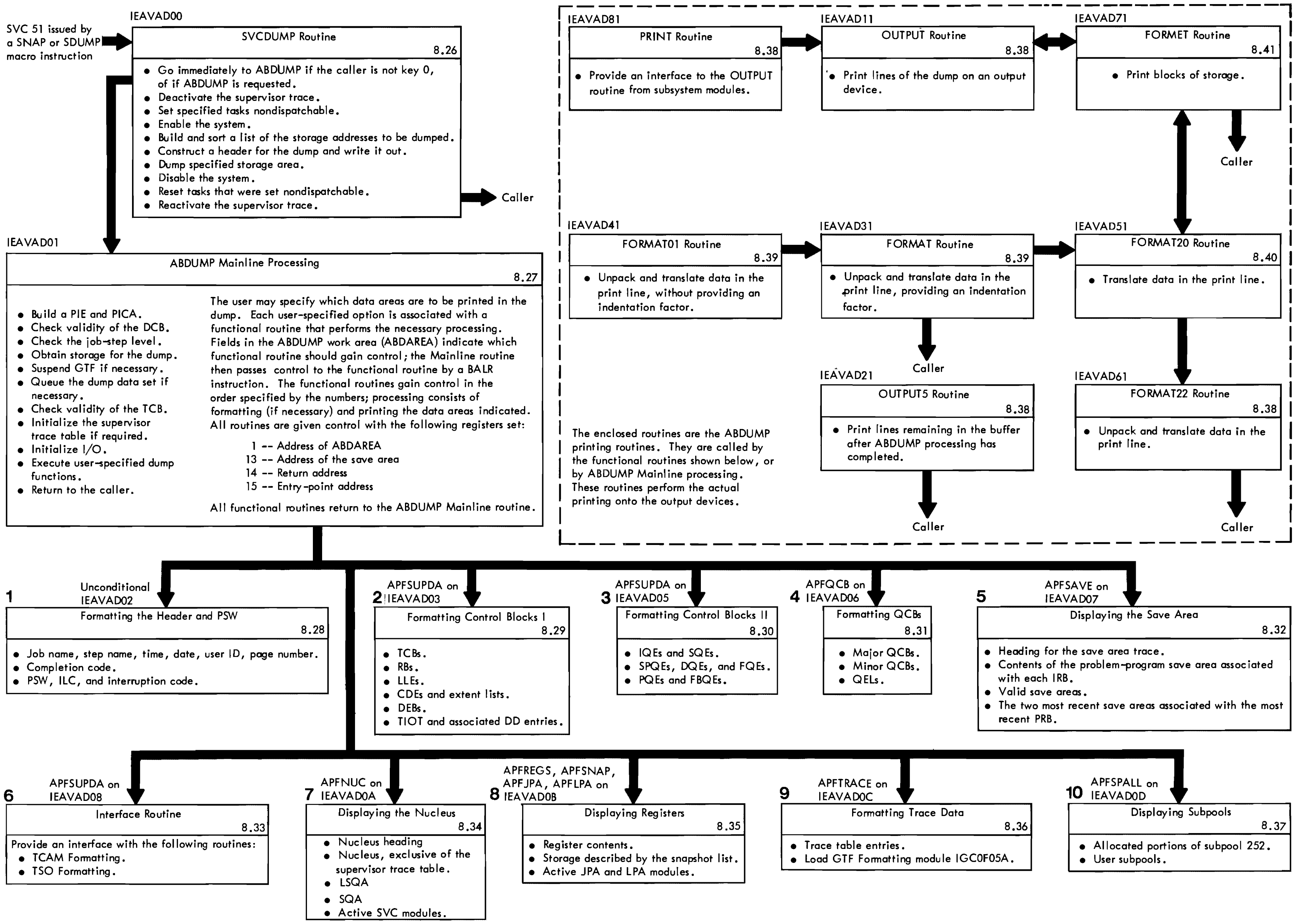
TABLE OF VALID ABEND RECURSIONS

TCBABCUR	Routine that failed	Processing that failed
X'01'	Initial Housekeeping Phase	WTO processing to inform the operator of the failure of the task while it owned the supervisor lock.
X'11'	Open Phase	OPEN processing for the dump data set.
X'12'	ABDUMP Phase	ABDUMP processing.
X'13'	Open Phase	WTO processing to inform the programmer of a failure during OPEN processing for the data set.
X'14'	ABDUMP Phase	WTO processing to inform the programmer of the failure to provide an abnormal termination dump.
X'15'	Mainline ABEND	WTP processing to inform the programmer of all entries in the type-1 SVC message table (MSGPHASE subroutine).
X'21'	Close Phase	I/O error received in the release of a shared DASD device.
X'22'	Close Phase	WTO processing to inform the operator of the failure of the task while it owned the supervisor lock.
X'23'	Close Phase	TCAM processing.
X'24'	Close Phase	SVC Dump failed.
X'25'	Close Phase	WTO indicating SVC Dump Failure failed.
X'31'	Must-Complete Phase	WTO processing to inform the programmer that the task failed while in system or step must-complete status.
X'32'	Must-Complete Phase	WTO processing to inform the programmer that a must-complete resource was released since it was defined as a step resource instead of a system resource.
X'33'	Must-Complete Phase	WTO processing to inform the programmer that even though a task is in must-complete status, there are no more must-complete resources allocated to the task.
X'34'	Must-Complete Phase	SVCDUMP processing (CRITPURG subroutine).
X'35'	Must-Complete Phase	WTO processing to inform the programmer of the failure of a SVCDUMP to provide a dump.
X'36'	Must-Complete Phase	WTO/WTOR processing to determine whether a must-complete resource is critical (ENQMSG subroutine).
X'41'	Close Phase	CLOSE processing for user data sets.
X'42'	Close Phase	WTO processing to inform the programmer of the failure to close data sets.

ROUTINE NAME: Recur-sion Phase

NOTES			ROUTINE NAME	LABEL
X'E1'	Critical Error Phase	Processing connected with one of the following routines failed: • Termination Interface routine -- purge of paging activity. • Purge I/O routine -- purge of I/O. • PRGQ subroutine -- purge of asynchronous exit queues. • WTOR Purge routine -- purge of WTOR requests. • ENQ/DEQ routine -- purge of queued resources. • WTO routine -- attempting to inform the operator of a task failure while owning the supervisor lock. • WTO routine -- attempting to inform the programmer that a task was halted. • HOOK processing for GTF.	Recursion Phase	

Diagram 8.25
Overview of the SVCDUMP and ABDUMP Routines



SVC 51 issued by a SNAP or SDUMP macro instruction

IEAVAD00
SVCDUMP Routine 8.26
- Go immediately to ABDUMP if the caller is not key 0, of if ABDUMP is requested.
- Deactivate the supervisor trace.
- Set specified tasks nondispatchable.
- Enable the system.
- Build and sort a list of the storage addresses to be dumped.
- Construct a header for the dump and write it out.
- Dump specified storage area.
- Disable the system.
- Reset tasks that were set nondispatchable.
- Reactivate the supervisor trace.

Caller

IEAVAD01
ABDUMP Mainline Processing 8.27
- Build a PIE and PICA.
- Check validity of the DCB.
- Check the job-step level.
- Obtain storage for the dump.
- Suspend GTF if necessary.
- Queue the dump data set if necessary.
- Check validity of the TCB.
- Initialize the supervisor trace table if required.
- Initialize I/O.
- Execute user-specified dump functions.
- Return to the caller.

The user may specify which data areas are to be printed in the dump. Each user-specified option is associated with a functional routine that performs the necessary processing. Fields in the ABDUMP work area (ABDAREA) indicate which functional routine should gain control; the Mainline routine then passes control to the functional routine by a BALR instruction. The functional routines gain control in the order specified by the numbers; processing consists of formatting (if necessary) and printing the data areas indicated. All routines are given control with the following registers set:

```
 1 -- Address of ABDAREA
13 -- Address of the save area
14 -- Return address
15 -- Entry-point address
```

All functional routines return to the ABDUMP Mainline routine.

IEAVAD81
PRINT Routine 8.38
- Provide an interface to the OUTPUT routine from subsystem modules.

IEAVAD11
OUTPUT Routine 8.38
- Print lines of the dump on an output device.

IEAVAD71
FORMET Routine 8.41
- Print blocks of storage.

Caller

IEAVAD41
FORMAT01 Routine 8.39
- Unpack and translate data in the print line, without providing an indentation factor.

IEAVAD31
FORMAT Routine 8.39
- Unpack and translate data in the print line, providing an indentation factor.

Caller

IEAVAD51
FORMAT20 Routine 8.40
- Translate data in the print line.

IEAVAD21
OUTPUT5 Routine 8.38
- Print lines remaining in the buffer after ABDUMP processing has completed.

Caller

IEAVAD61
FORMAT22 Routine 8.38
- Unpack and translate data in the print line.

Caller

The enclosed routines are the ABDUMP printing routines. They are called by the functional routines shown below, or by ABDUMP Mainline processing. These routines perform the actual printing onto the output devices.

1 Unconditional IEAVAD02
Formatting the Header and PSW 8.28
- Job name, step name, time, date, user ID, page number.
- Completion code.
- PSW, ILC, and interruption code.

2 APFSUPDA on IEAVAD03
Formatting Control Blocks I 8.29
- TCBs.
- RBs.
- LLEs.
- CDEs and extent lists.
- DEBs.
- TIOT and associated DD entries.

3 APFSUPDA on IEAVAD05
Formatting Control Blocks II 8.30
- IQEs and SQEs.
- SPQEs, DQEs, and FQEs.
- PQEs and FBQEs.

4 APFQCB on IEAVAD06
Formatting QCBs 8.31
- Major QCBs.
- Minor QCBs.
- QELs.

5 APFSAVE on IEAVAD07
Displaying the Save Area 8.32
- Heading for the save area trace.
- Contents of the problem-program save area associated with each IRB.
- Valid save areas.
- The two most recent save areas associated with the most recent PRB.

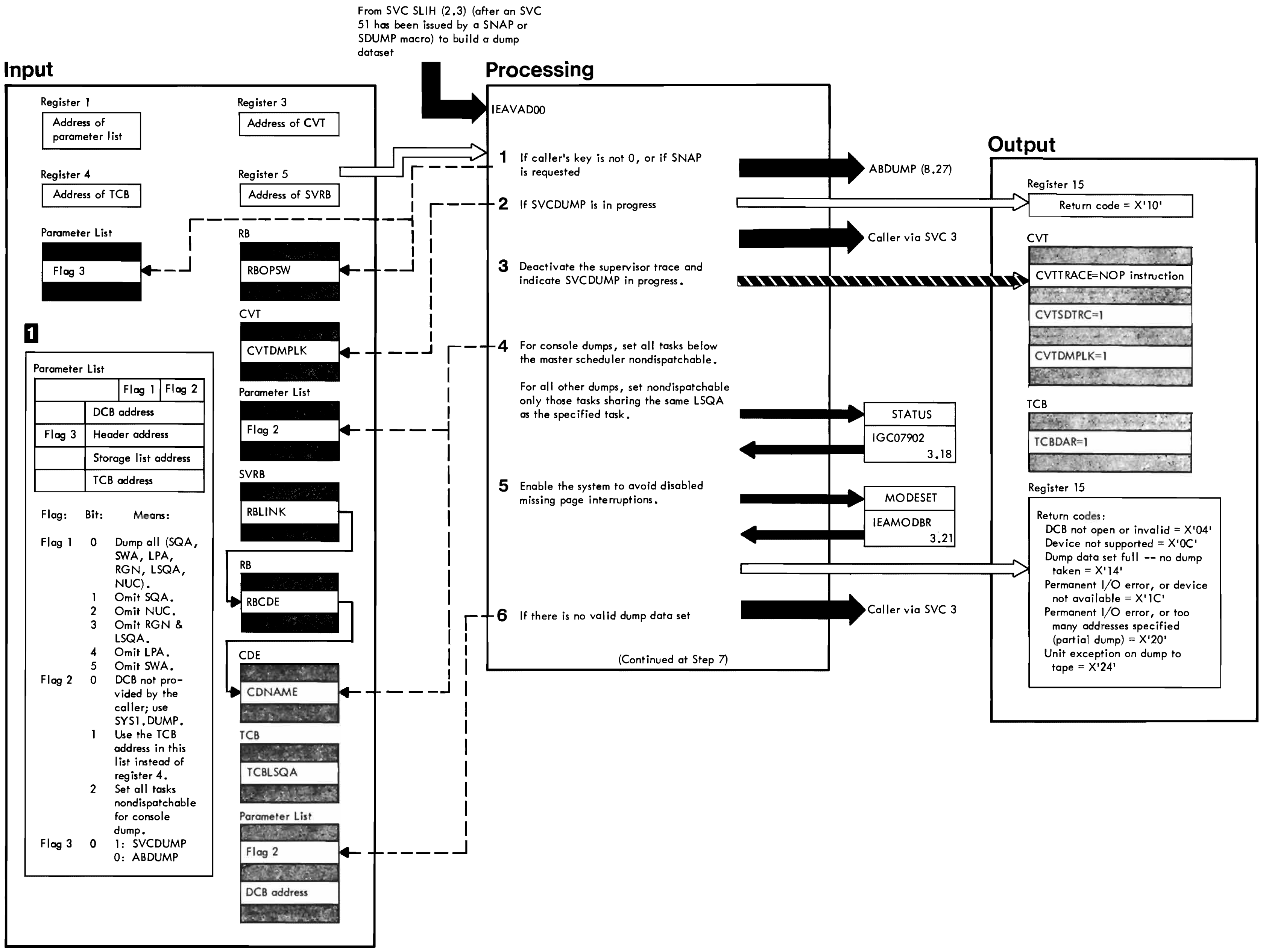
6 APFSUPDA on IEAVAD08
Interface Routine 8.33
Provide an interface with the following routines:
- TCAM Formatting.
- TSO Formatting.

7 APFNUC on IEAVAD0A
Displaying the Nucleus 8.34
- Nucleus heading
- Nucleus, exclusive of the supervisor trace table.
- LSQA
- SQA
- Active SVC modules.

8 APFREGS, APFSNAP, APFJPA, APFLPA on IEAVAD0B
Displaying Registers 8.35
- Register contents.
- Storage described by the snapshot list.
- Active JPA and LPA modules.

9 APFTRACE on IEAVAD0C
Formatting Trace Data 8.36
- Trace table entries.
- Load GTF Formatting module IGC0F05A.

10 APFSPALL on IEAVAD0D
Displaying Subpools 8.37
- Allocated portions of subpool 252.
- User subpools.



Flag:	Bit:	Means:
Flag 1	0	Dump all (SQA, SWA, LPA, RGN, LSQA, NUC).
	1	Omit SQA.
	2	Omit NUC.
	3	Omit RGN & LSQA.
	4	Omit LPA.
	5	Omit SWA.
Flag 2	0	DCB not provided by the caller; use SYS1.DUMP.
	1	Use the TCB address in this list instead of register 4.
	2	Set all tasks nondispatchable for console dump.
Flag 3	0	1: SVCDUMP 0: ABDUMP

1 If caller's key is not 0, or if SNAP is requested

2 If SVCDUMP is in progress

3 Deactivate the supervisor trace and indicate SVCDUMP in progress.

4 For console dumps, set all tasks below the master scheduler nondispatchable.

For all other dumps, set nondispatchable only those tasks sharing the same LSQA as the specified task.

5 Enable the system to avoid disabled missing page interruptions.

6 If there is no valid dump data set

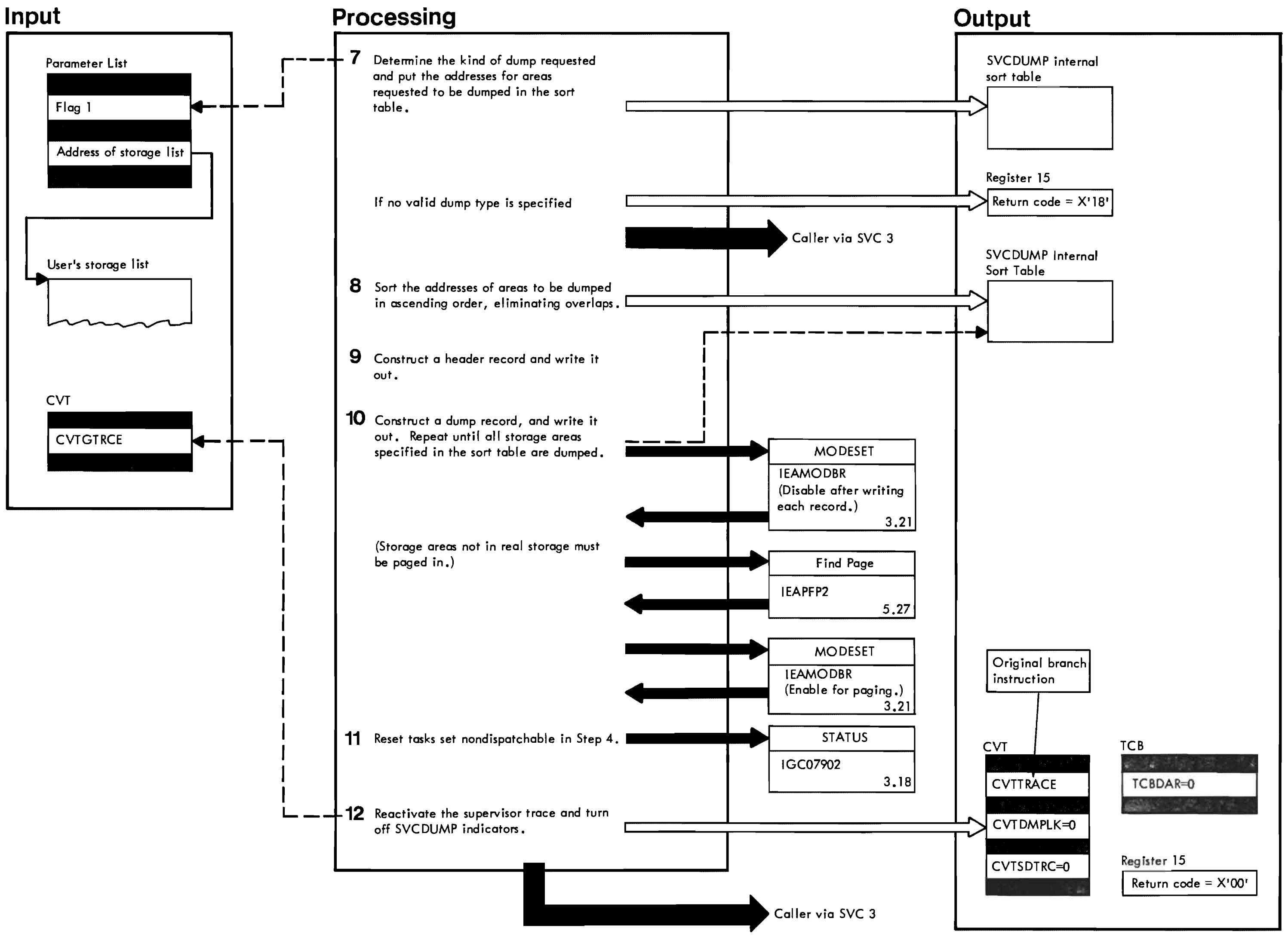
(Continued at Step 7)

Return codes:
- DCB not open or invalid = X'04'
- Device not supported = X'0C'
- Dump data set full -- no dump taken = X'14'
- Permanent I/O error, or device not available = X'1C'
- Permanent I/O error, or too many addresses specified (partial dump) = X'20'
- Unit exception on dump to tape = X'24'

Diagram 8.26 (Steps 1-6) SVCDUMP Routine (Module IEAVAD00)

NOTES	ROUTINE NAME	LABEL
1 Before referencing the parameter list, it is checked to determine whether it is in real storage. If it is not, to avoid paging errors, MODESET is called to enable the system. The parameter list is referenced to have it paged in; MODESET is called again to disable the system.	SVCDUMP	DMPLRA
1 RBOPSW contains the caller's key. Only key 0 callers can use SVCDUMP; all others must use ABDUMP.		DMPSNAPP
2 CVTDMPLK, when set, means that SVCDUMP is in progress.		
3 CVTSDTRC is set to indicate that SVCDUMP has deactivated the trace. CVTTRACE contains a branch instruction; to deactivate the trace, a NOP instruction is inserted. TCBDAR, when set, means that SVCDUMP is in progress for this TCB.		
4 If Flag 2 indicates a console dump, CDNAME is checked. It must contain "IEE60110" to prove that the caller is the Console Dump routine.		DMPTLSQA
5		DMPCKDC
6 The dump data set is invalid under these conditions: • It cannot be found. • It is not located on a tape device nor a supported direct access device (SVCDUMP does not support devices 2302, 2303, 2311, or 2321). • The device is offline . • The data set is not open. • The data set is full.		DMPDCBLK

Diagram 8.26 (Steps 7-12)
SVCDUMP Routine



7 Determine the kind of dump requested and put the addresses for areas requested to be dumped in the sort table.

If no valid dump type is specified

8 Sort the addresses of areas to be dumped in ascending order, eliminating overlaps.

9 Construct a header record and write it out.

10 Construct a dump record, and write it out. Repeat until all storage areas specified in the sort table are dumped.

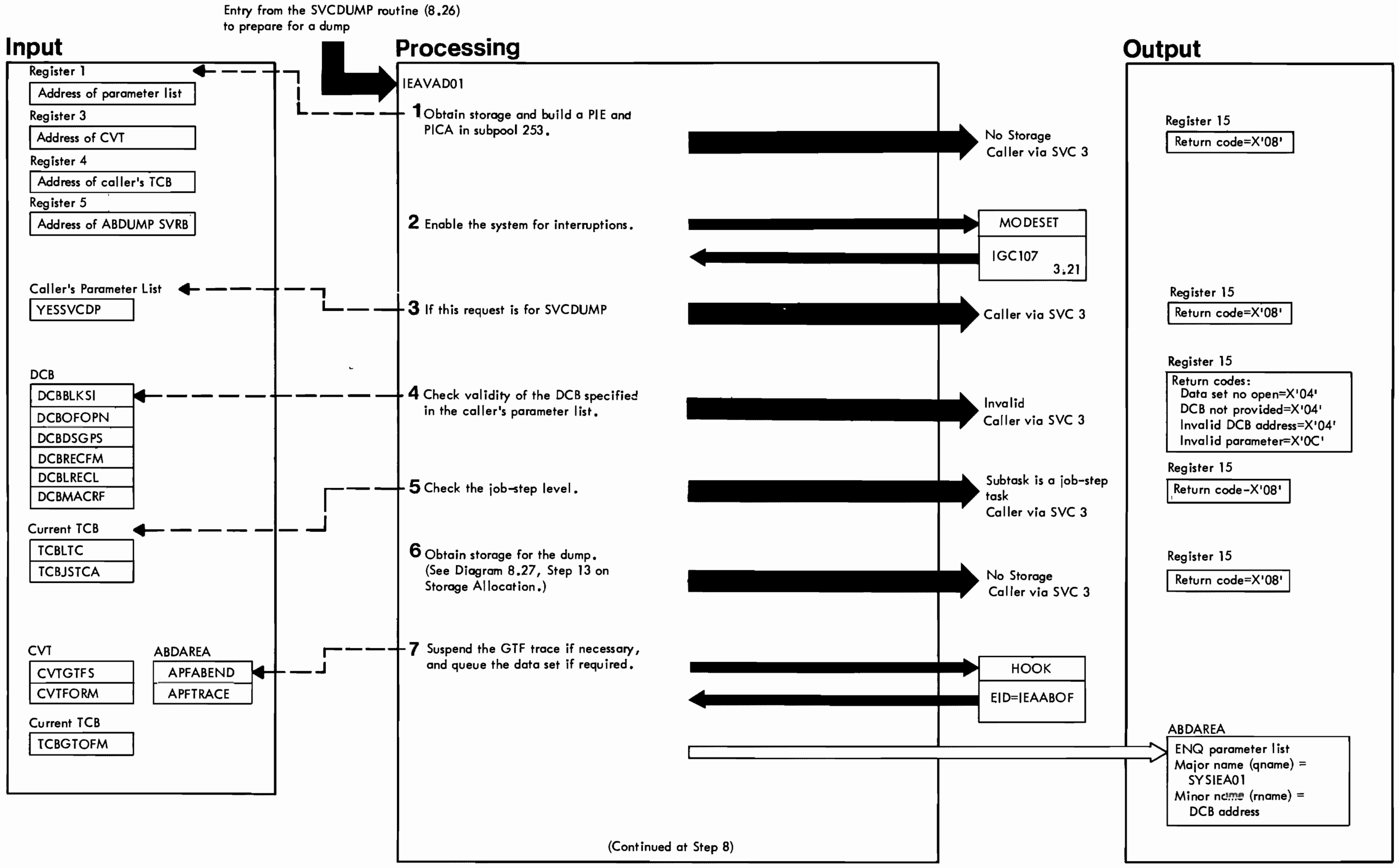
(Storage areas not in real storage must be paged in.)

11 Reset tasks set nondispatchable in Step 4.

12 Reactivate the supervisor trace and turn off SVCDUMP indicators.

Diagram 8.26 (Steps 7-12) SVCDUMP Routine (Module IEAVAD00)

NOTES	ROUTINE NAME	LABEL
7 At least one of these conditions must specify the dump type: • "Dump-all-storage" request (see the parameter list inset) • Storage list address is not 0, but it points to a list of addresses to be dumped • TCB is specified in the parameter list	SVCDUMP	DMPINIT DMPCKPL
8 Once the STATUS routine has been entered to set tasks nondispatchable, any error exit must first reset tasks dispatchable for normal completion.		DMPSORT
9		DMPSETHD
10		DMPADINC
11		DMPERET
12 If CVTGTRCE=1, GTF has set the supervisor trace nondispatchable. In this case, CVTTRACE is not reset.		



Entry from the SVCDUMP routine (8.26) to prepare for a dump

Input

Register 1
Address of parameter list
Register 3
Address of CVT
Register 4
Address of caller's TCB
Register 5
Address of ABDUMP SVRB

Caller's Parameter List
YESSVCDP

DCB
DCBBLKSI
DCBOFOPN
DCBDSGPS
DCBRECFM
DCBLRECL
DCBMACRF

Current TCB
TCBLTC
TCBJSTCA

CVT
CVTGTFS
CVTFORM

ABDAREA
APFABEND
APFTRACE

Current TCB
TCBGTOFM

Processing

IEAVAD01

1 Obtain storage and build a PIE and PICA in subpool 253.

No Storage
Caller via SVC 3

2 Enable the system for interruptions.

MODESET
IGC107 3.21

3 If this request is for SVCDUMP

Caller via SVC 3

4 Check validity of the DCB specified in the caller's parameter list.

Invalid
Caller via SVC 3

5 Check the job-step level.

Subtask is a job-step task
Caller via SVC 3

6 Obtain storage for the dump. (See Diagram 8.27, Step 13 on Storage Allocation.)

No Storage
Caller via SVC 3

7 Suspend the GTF trace if necessary, and queue the data set if required.

HOOK
EID=IEAABOF

(Continued at Step 8)

Output

Register 15
Return code=X'08'

Register 15
Return code=X'08'

Register 15
Return codes:
Data set no open=X'04'
DCB not provided=X'04'
Invalid DCB address=X'04'
Invalid parameter=X'0C'

Register 15
Return code-X'08'

Register 15
Return code=X'08'

ABDAREA
ENQ parameter list
Major name (qname) = SYSIEA01
Minor name (rname) = DCB address

Diagram 8.27 (Steps 1-7) ABDUMP Mainline Processing (Module IEAVAD01)

NOTES	ROUTINE NAME	LABEL
1 If storage is allocated, a PIE, which points to a PICA, is built. The PICA accepts only protection program interruptions (code 4). INITX1 is specified as the ABDUMP exit routine; this routine gains control if an invalid missing page interrruption (a page that is unassigned is referred to) occurs. The RBUPR flag in the RB is set to 1 to indicate that a PIE exists.	IEAVAD01	
4 The DCB is valid if the following values are specified: • DCBOFOPN = 1 (DCB open). • DCBDSGPS = 1 (physical sequential organization). • DCBRECFM = X'54' (variable blocked format with ASCII control characters). • DCBLRECL = 125 (logical record length). • DCBBLKSI = 882 or 1632 (block size). • DCBMACRF = X'0020' (data set open for BSAM Writes). If an invalid missing page interruption occurs during the validity check of the DCB, the exit routine INITX1 is called to restore registers 14-2 from the PIE. For an invalid missing page interruption, or any other invalid DCB condition, all storage acquired is freed before passing control to the caller.		
5 Because processing is limited to tasks that do not have job-step tasks as their descendants, a test is made (while the system is disabled) to ensure that the current task does not have a subtask that is also a job-step task. If no such subtasks exist, asynchronous exits (TCBFX=1) and invalid missing page interruption processing (RBUPR=0) are prohibited. Processing continues with the allocation of storage for the dump.		
7 If ABEND is not the caller, and the user-requested trace data is displayed, tests are made to determine whether GTF is active in the system (CVTGTFS), and whether "in-core" tracing (CVTFORM) has been specified. If so, tracing is suspended now if not done earlier. If ABEND is not the caller, an ENQ parameter list is built, and an exclusive ENQ macro instruction is issued for the data set.		DOENQ

Diagram 8.27 (Steps 8-12)
ABDUMP Mainline Processing

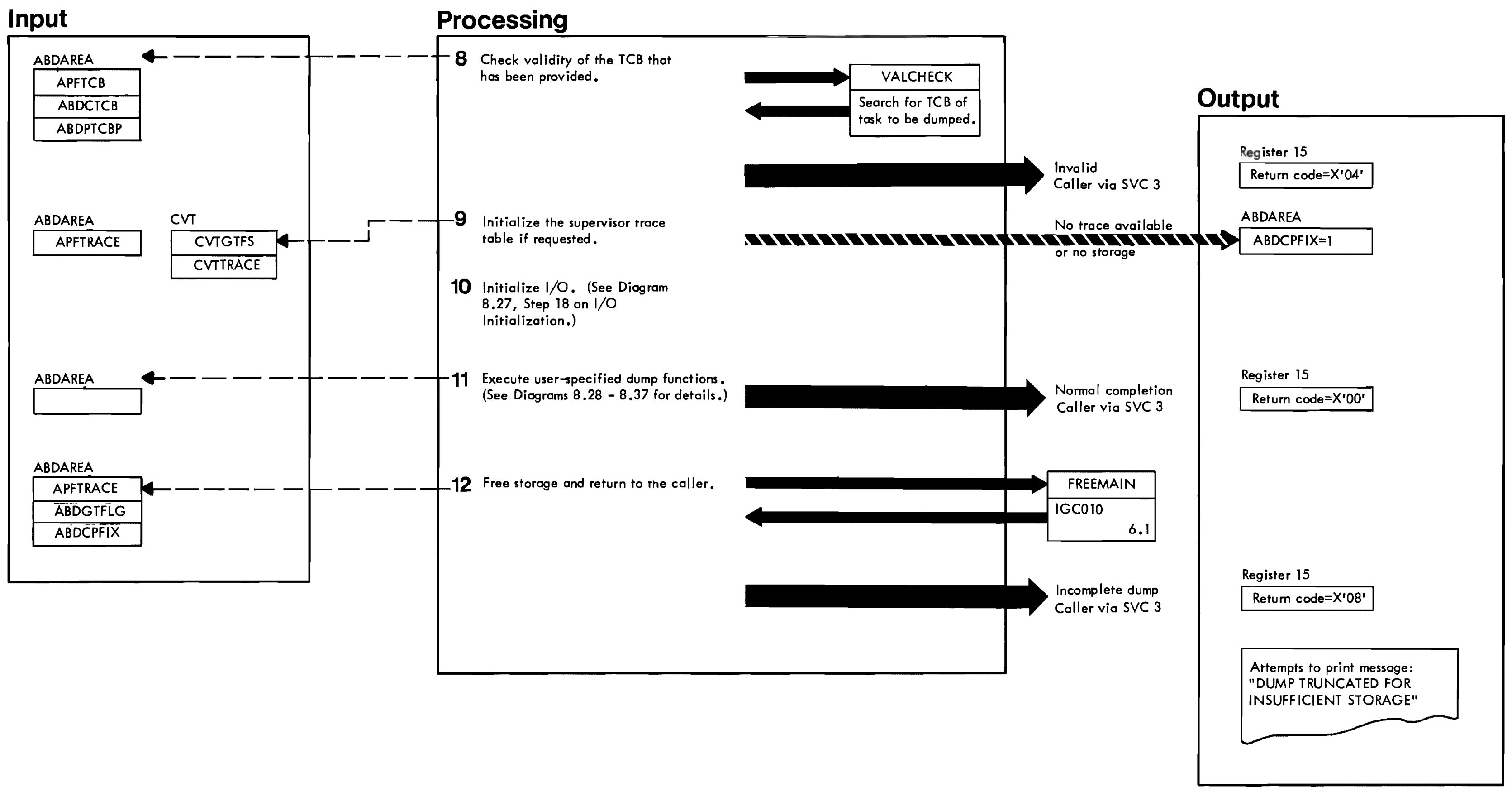
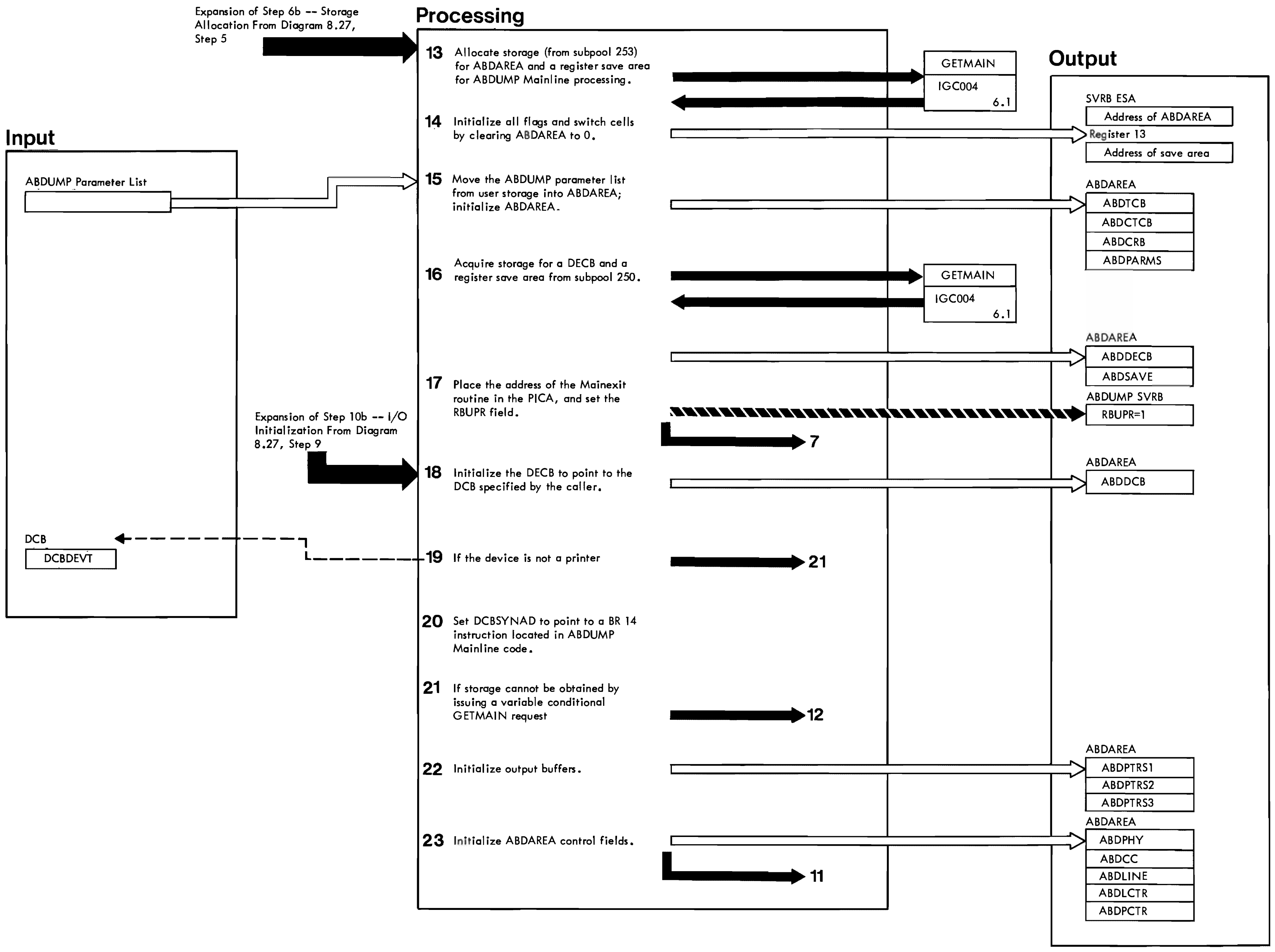

Input
ABDAREA
APFTCB
ABDCTCB
ABDPTCBP
ABDAREA
APFTRACE
CVT
CVTGTFS
CVTTRACE
ABDAREA
ABDAREA
APFTRACE
ABDGTFLG
ABDCPFIX
Processing
8 Check validity of the TCB that has been provided.
VALCHECK
Search for TCB of task to be dumped.
Invalid
Caller via SVC 3
9 Initialize the supervisor trace table if requested.
No trace available or no storage
10 Initialize I/O. (See Diagram 8.27, Step 18 on I/O Initialization.)
11 Execute user-specified dump functions. (See Diagrams 8.28 - 8.37 for details.)
Normal completion
Caller via SVC 3
12 Free storage and return to the caller.
FREEMAIN
IGC010
6.1
Incomplete dump
Caller via SVC 3
Output
Register 15
Return code=X'04'
ABDAREA
ABDCPFIX=1
Register 15
Return code=X'00'
Register 15
Return code=X'08'
Attempts to print message: "DUMP TRUNCATED FOR INSUFFICIENT STORAGE"

Diagram 8.27 (Steps 8-12) ABDUMP Mainline Processing (Module IEAVAD01)

NOTES	ROUTINE NAME	LABEL
8 Under the following circumstances, the dump is considered to be that of the current task; further checking of the TCB is bypassed: • User did not specify to dump another task. • User-specified dump is not of the current task, but the TCB address (ABDPTCBP) is set to 0 or to the address of the current task's TCB (ABDCTCB).	IEAVAD01	NOENQ
The address of the TCB for the task to be dumped (ABDTCB) is set to the address of the current task's TCB (ABDCTCB), and execution continues with the initialization of the supervisor trace table (Step 9). If the current task is not being dumped, a check is made to ensure that the task to be dumped is in the same job step as the task which requested the dump. Before checking the TCB address supplied by the user, all tasks in the job step except the current task are set nondispatchable. If the VALCHECK subroutine indicates a valid TCB, the ABDTCB field is set to the address of the user-supplied TCB (ABDPTCBP). Execution continues with Step 9. If the TCB address is invalid, a HOOK macro instruction is issued to restart GTF (if suspended earlier). The job step is set dispatchable (using STATUS), and the dump data set is dequeued if necessary. After freeing storage, return is made to the caller.		DUMPCUR EXIT1 FREEPIE
9 If a supervisor trace table dump has been requested, an attempt is made to acquire storage (from subpool 252) in which to move the table. If storage is allocated, the beginning (ABDFP) and ending (ABDLP) addresses of the buffer are saved in ABDAREA. The current trace table entry is found by adding the displacement from the beginning of the table to the initial address of the buffer. This address is placed in ABDCP1. The trace table is moved, and the current trace table entry is found as previously. This new address is stored in ABDCP. The addresses of the two entries are used later by the Formatting Trace Data routine (IEAVAD0C). The entries existing between ABDCP1 and ABDCP are ignored because they occurred during the time the trace table was being moved. If no trace is available, or there is insufficient storage, IEAVAD0C prints the message: "TRACE NOT AVAILABLE." Note: The trace table dump is provided only in a SNAP dump.		VALIDTCB

NOTES	ROUTINE NAME	LABEL
11 The functional areas are given control in the order described in Diagram 8.25.	IEAVAD01	
12 If a failure occurred before IEAVAD0C was invoked, ABDUMP Mainline must perform two checks: • If GTF was suspended earlier (ABDGTFLG), it must be restarted by issuing the HOOK macro instruction. • Otherwise, if a trace table buffer was obtained (ABDCPFIX), it is freed.		TRACELN
If a failure occurred in IEAVAD0C processing, the return code indicates whether cleanup processing is required or was performed by IEAVAD0C.		
If the dump has been completed without error, the "END OF DUMP" message is printed and the contents of the blocking buffers are written out. The job step is set dispatchable, if necessary, and a DEQ macro instruction is issued if the ENQ macro instruction was issued earlier. The following storage areas are freed: blocking buffers, save areas, ABDAREA, and PIE/PICA storage. Note: The buffers are printed by the OUTPUT (IEAVAD11) and OUTPUT5 (IEAVAD21) routines.		EXIT2A EXIT1 FREEAREA FREEPIE

Diagram 8.27 (Steps 13-23) ABDUMP Mainline Processing



Input

ABDUMP Parameter List

DCB

DCBDEVT

Expansion of Step 6b -- Storage Allocation From Diagram 8.27, Step 5

Expansion of Step 10b -- I/O Initialization From Diagram 8.27, Step 9

Processing

13 Allocate storage (from subpool 253) for ABDAREA and a register save area for ABDUMP Mainline processing.

14 Initialize all flags and switch cells by clearing ABDAREA to 0.

15 Move the ABDUMP parameter list from user storage into ABDAREA; initialize ABDAREA.

16 Acquire storage for a DECB and a register save area from subpool 250.

17 Place the address of the Mainexit routine in the PICA, and set the RBUPR field.

18 Initialize the DECB to point to the DCB specified by the caller.

19 If the device is not a printer

20 Set DCBSYNAD to point to a BR 14 instruction located in ABDUMP Mainline code.

21 If storage cannot be obtained by issuing a variable conditional GETMAIN request

22 Initialize output buffers.

23 Initialize ABDAREA control fields.

Output

SVRB ESA

Address of ABDAREA

Register 13

Address of save area

ABDAREA

ABDTCB

ABDCTCB

ABDCRB

ABDPARMS

ABDAREA

ABDDECB

ABDSAVE

ABDUMP SVRB

RBUPR=1

ABDAREA

ABDDCB

ABDAREA

ABDPTRS1

ABDPTRS2

ABDPTRS3

ABDAREA

ABDPHY

ABDCC

ABDLINE

ABDLCTR

ABDPCTR

Diagram 8.27 (Steps 13-23) ABDUMP Mainline Processing (Module IEAVAD01)

NOTES	ROUTINE NAME	LABEL
Storage Allocation 13	IEAVAD01	

15 The ABDUMP parameter list is moved into the ABDUMP work area (ABDAREA) to enable the list to be examined without encountering missing page interruptions. The following fields are initialized:
ABDTCB = address of TCB dumped
ABDCTCB = address of current TCB
ABDCRB = address of ABDUMP SVRB
ABDPARMS = ABDUMP parameter list		

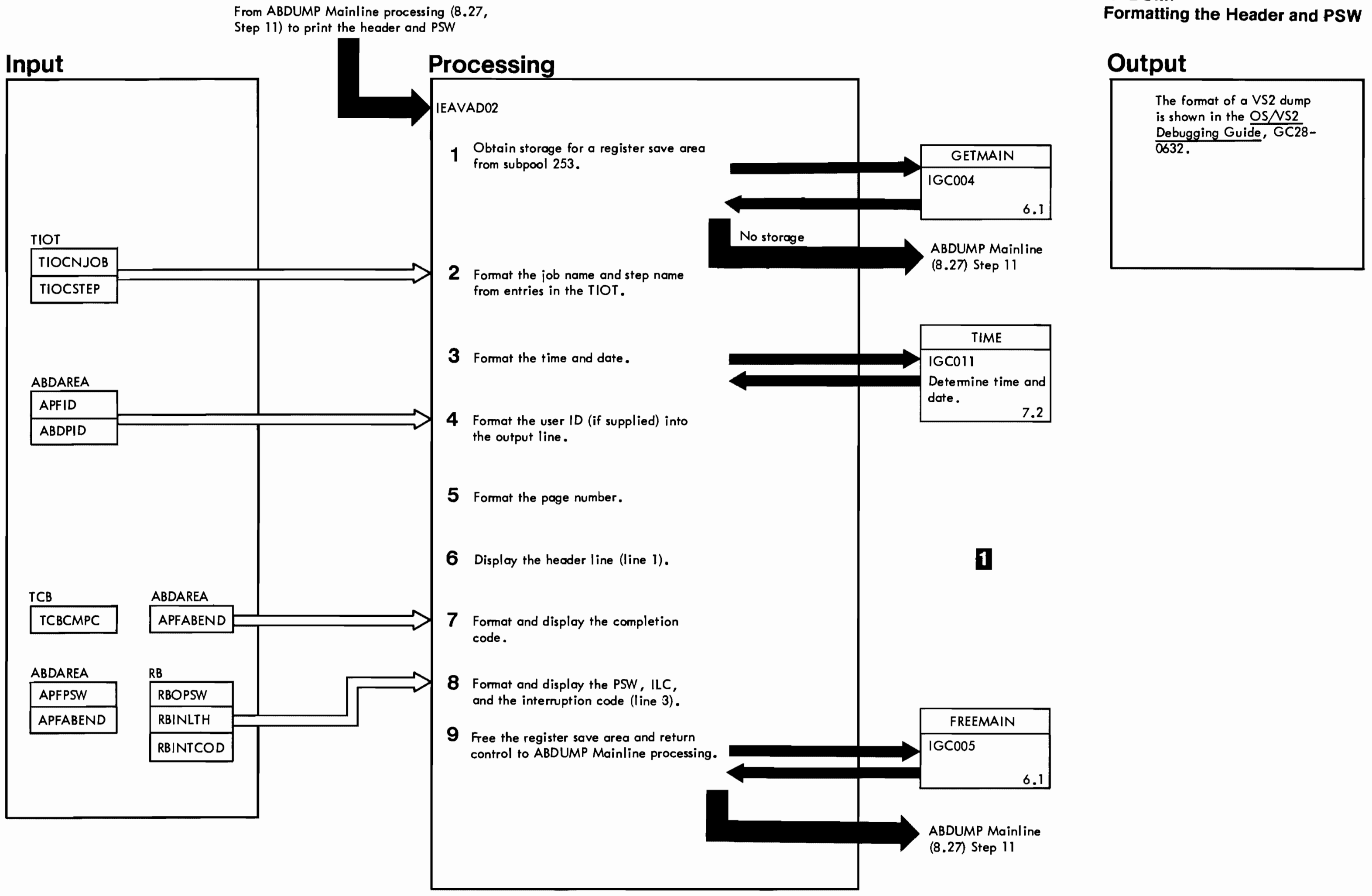
16 The save area is later used by the OUTPUT (IEAVAD11) subroutine. If the storage cannot be allocated, all storage obtained earlier must be freed.		
17 The exit routine INITX1 is replaced with the Mainexit routine. The RBUPR field is set to 1 to indicate that the Mainexit routine gains control when an invalid missing page interruption occurs. The Mainexit routine issues an SVC 13 instruction to pass control to ABEND (with a X'0C4' completion code) if an exit routine was not specified. If an exit routine to handle invalid missing page interruptions was specified (in field ABDUPRXT), this routine is given control.		

NOTES	ROUTINE NAME	LABEL
I/O Initialization 18	IEAVAD01	DONETRCE
19 The dump can be printed (one line at a time) on a printer, or written (as a block of storage) on tape or a direct access device. If the device type field (DCBDEVT) contains X'48', the device is a printer.		
20 By pointing to a BR 14 instruction, print errors are ignored and the dump continues, with output unblocked.		
21 The GETMAIN macro instruction is issued for a minimum of 256 bytes or a maximum indicated by the DCBBLKSI field.		

22 The output buffer pointers are initialized to the following values:
ABDPTRS1 = initial buffer address
ABDPTRS2 = next available buffer address
ABDPTRS3 = end of buffer + 1		

23 The ABDAREA control fields are initialized to the following values: ABDPHY (record length) = 129 ABDCC (ASCII control character) = blank ABDLINE (print line) = blank ABDLCTR (line count) = 1 ABDPCTR (page count) = 1		SETRECLG

Diagram 8.28
ABDUMP—
Formatting the Header and PSW



1 Obtain storage for a register save area from subpool 253.

2 Format the job name and step name from entries in the TIOT.

3 Format the time and date.

4 Format the user ID (if supplied) into the output line.

5 Format the page number.

6 Display the header line (line 1).

7 Format and display the completion code.

8 Format and display the PSW, ILC, and the interruption code (line 3).

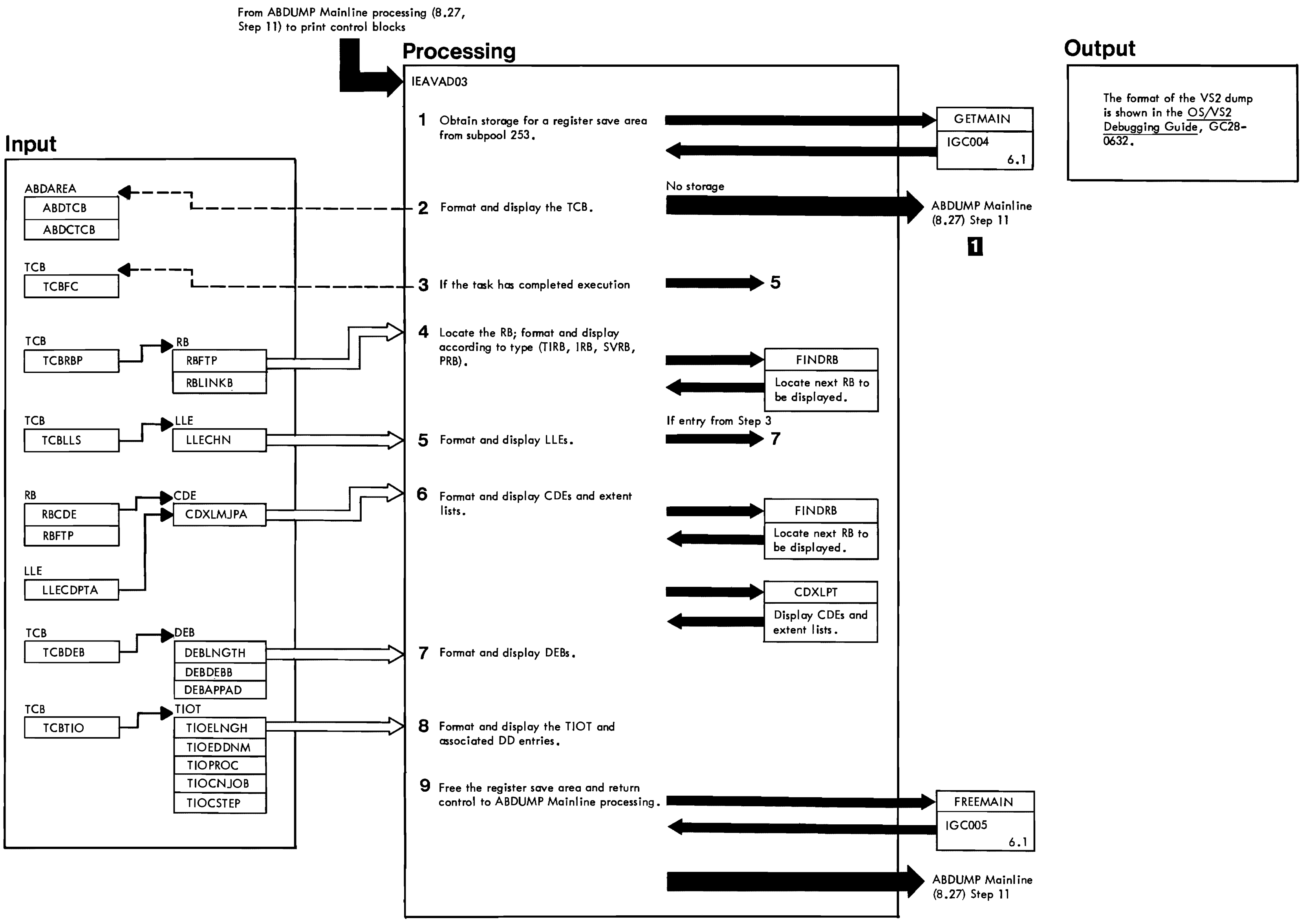
9 Free the register save area and return control to ABDUMP Mainline processing.

The format of a VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.28 ABDUMP -- Formatting the Header and PSW (Module IEAVAD02)

NOTES	ROUTINE NAME	LABEL
1 If the GETMAIN request fails, control returns to ABDUMP Mainline processing with a return code of X'04'.	IEAVAD02	
3 The TIME macro instruction is issued to obtain the correct time and date.		
5 The page number (found in field PAGEMSG) is initialized to C'0001'.		
7 If ABEND is the caller (APFABEND=0), the completion code is formatted and displayed. If the system completion code is not 0, it is used; otherwise the user code is displayed. No completion code is displayed if the dump is not an abnormal (ABEND) dump.		
8 The APFPSW field, if set, indicates that the PSW, ILC (interruption length code), and interruption code are formatted and displayed from the RB of the task in control at the time of the SNAP or ABEND (APFABEND).		
9 A return code is passed in register 15: X'00' - successful completion X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine		FREERET
1 The following ABDUMP print routines are called: • FORMAT (IEAVAD31) Unpacks and translates data in the print line, providing an indentation factor. Called to execute Steps 3, 7, and 8. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute Steps 6, 7, and 8. Should either ABDUMP print routine fail (return code ≠ X'00'), control passes to cleanup processing (Step 9). Note: The appropriate label precedes each field. For example: JOB jobname STEP stepname		

**Diagram 8.29
ABDUMP—
Formatting Control Blocks I**



1 Obtain storage for a register save area from subpool 253.

2 Format and display the TCB.

3 If the task has completed execution

4 Locate the RB; format and display according to type (TIRB, IRB, SVRB, PRB).

5 Format and display LLEs.

6 Format and display CDEs and extent lists.

7 Format and display DEBs.

8 Format and display the TIOT and associated DD entries.

9 Free the register save area and return control to ABDUMP Mainline processing.

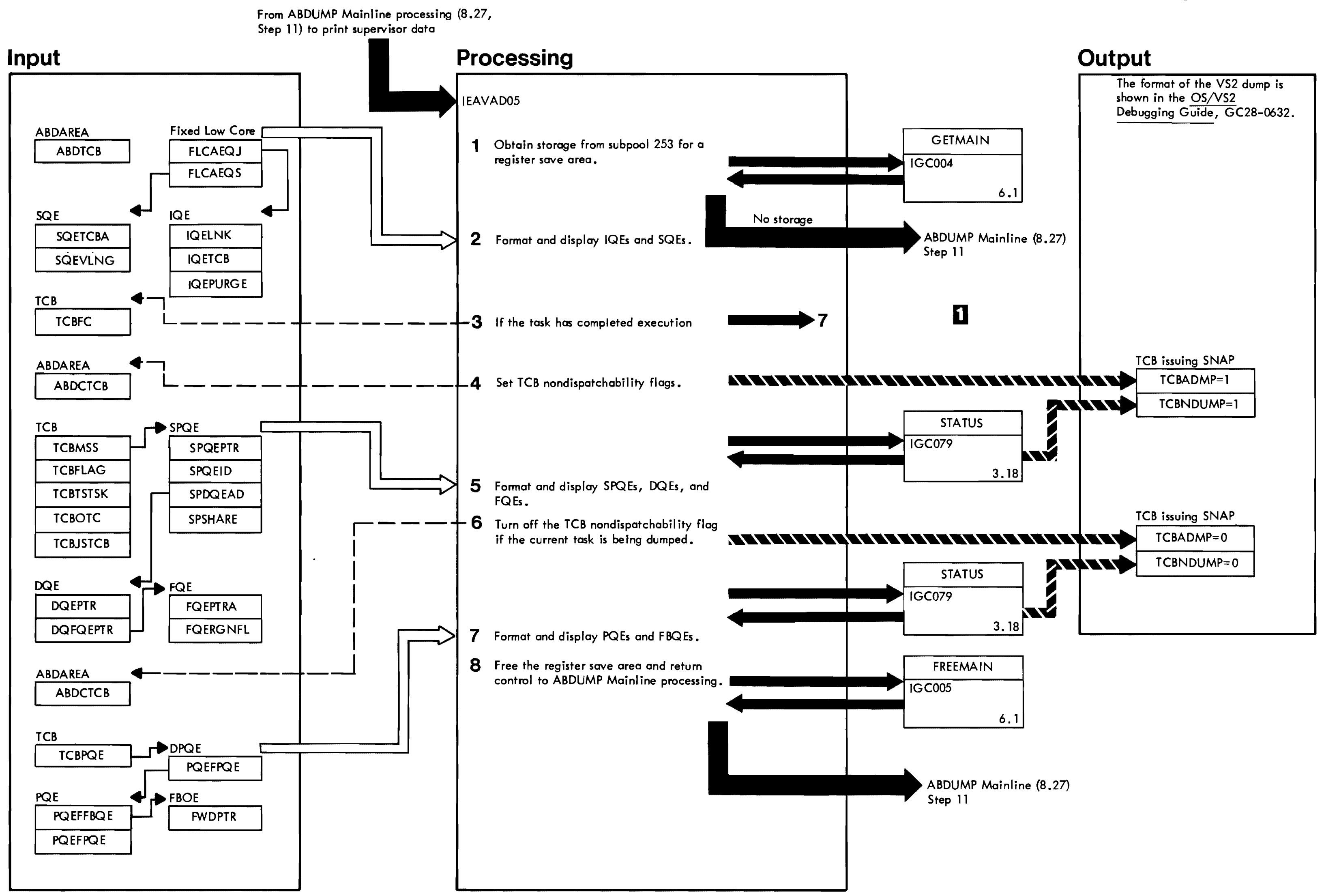
The format of the VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.29 ABDUMP -- Formatting Control Blocks I (Module IEAVAD03)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD03	
2 The TCB is formatted and displayed in three sections: • Start of the TCB to the start of the TCB register save area. • TCB register save area -- this section is displayed only if the TCB to be displayed (ABDTCB) is not the current TCB (ABDCTCB). • End of the TCB register save area to the end of the TCB.		TCBOUT
4 The FINDRB subroutine is entered to find the previous RB on the task queue. The RB at the beginning of the queue, the oldest RB, is displayed first. The last RB placed on the task queue, the youngest, is displayed last.		RBOUT
5 The LLEs are displayed three to a line in the order in which they appear on the task queue.		LLEOUT
6 The FINDRB subroutine is invoked to locate CDEs associated with RBs. The CDEs are formatted and displayed in the same order as the RBs. After the RB queue has been exhausted, any remaining CDEs associated with LLEs are formatted and displayed. Upon completion, the extent lists associated with CDEs are formatted and displayed in the same order as the CDEs.		CDEOUT GETRB LLECDE
7 The DEBs are formatted and displayed in the order in which they appear on the task queue. The DEB I/O AVT (appendage vector table) is displayed in a separate block if it does not appear in front of the DEB. Because the system is enabled for interruptions during formatting and display processing, an exit is established in the FORMET routine (IEAVAD71) to handle missing pages due to DEBs freed during I/O operations. Because the DEB pointer can be changed during I/O, a count is kept of the number of DEBs that have been displayed. The DEB queue is searched while the system is disabled. If the exit is taken, a message is printed stating that storage is invalid; the dump is not truncated.		DEBOUT

NOTES	ROUTINE NAME	LABEL
8 The fixed portion of the TIOT is formatted and displayed first. The DD entries (beginning with the first byte after the fixed portion) are then formatted and displayed. This processing continues until the null entry is found.	IEAVAD03	TIOTOUT
9 A return code is passed in register 15: X'00' - successful completion. X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine.		FREERET
1 The following ABDUMP print routines are called: • FORMAT (IEAVAD31) Unpacks and translates data in the print line, providing an indentation factor. Called to execute Steps 2, 4, 5, and 6. • FORMAT22 (IEAVAD61) Unpacks and translates data in the print line. Called to execute Steps 4, 6, and 8. • FORMET (IEAVAD71) Prints block of storage. Called to execute Step 7. • OUTPUT (IEAVAD11) Prints lines of the dump on the output device. Called to execute Steps 2, 4, 5, 6, 7, and 8. Should any of the ABDUMP print routines fail (return code ≠ X'00'), control passes to cleanup processing (Step 9). Note: The appropriate label precedes each field.		

Diagram 8.30 ABDUMP— Formatting Control Blocks II



IEAVAD05

1 Obtain storage from subpool 253 for a register save area.

2 Format and display IQEs and SQEs.

3 If the task has completed execution

4 Set TCB nondispatchability flags.

5 Format and display SPQEs, DQEs, and FQEs.

6 Turn off the TCB nondispatchability flag if the current task is being dumped.

7 Format and display PQEs and FBQEs.

8 Free the register save area and return control to ABDUMP Mainline processing.

The format of the VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.30 ABDUMP -- Formatting Control Blocks II (Module IEAVAD05)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD05	
2 The IQEs and SQEs are formatted and displayed in the same manner; the IQEs are processed first. After the IQE queue has been exhausted (the high-order byte in the last IQE contains an X'FF'), the loop is initialized for the SQE search.		SQEOUT
Under the following conditions, the IQE/SQE is formatted and displayed: • The purge flag (IQEPURGE/SQEPURGE) is on -- the element is not freed until it has been displayed. • The TCB address associated with the element (IQETCB/SQETCBA) is identical to the TCB address of the task being displayed (ABDTCB).		IQESQELP
If either of these conditions does not exist the IQE/SQE is ignored, and the search continues for the next IQE/SQE.		
Note: The search occurs in the disabled state with the system enabled. The formatting and displaying occurs with the system enabled.		IQEDISLP
3		VMMOUT
4 If the current task is being dumped, all tasks in the job step (except the current task) are set nondispatchable. This is done to ensure the reliability of the virtual storage supervision (VSS) control blocks.		
5 The virtual storage supervision control blocks are formatted and displayed in the order: SPQE, DQE, FQE. If the subpool is shared, the shared and owner SPQEs and their associated DQEs and FQEs are formatted and displayed if one of the following conditions exists:		NEXTSPQE
• The task being dumped is not a key 0 nor a TSO task. • The owner SPQE is under a task within the job step of the task being dumped.		FMSPQE OUTSLOOP

NOTES	ROUTINE NAME	LABEL
6	IEAVAD05	DISPACHK
7 The dummy PQE is first formatted and displayed. Then the PQE and the associated FBQE are processed. If the FBQE address is identical to the PQE, the end of the FBQE queue has been reached and the next PQE is searched.		PQEOUT NEXTPQE GETFBQE
8 A return code is passed in register 15: X'00' - successful completion. X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine.		FREERET
1 The following ABDUMP print routines are called: • FORMAT22 (IEAVAD61) Unpacks and translates data in the print line. Called to execute Step 5. • FORMAT (IEAVAD31) Unpacks and translates data in the print line, providing an indentation factor. Called to execute Steps 2 and 7. • OUTPUT (IEAVAD11) Prints line of the dump on an output device. Called to execute Steps 2, 5, and 7. Should any of the ABDUMP print routines fail (return code ≠ '00'), control passes to cleanup processing (Step 8). Note: The appropriate label precedes each field.		

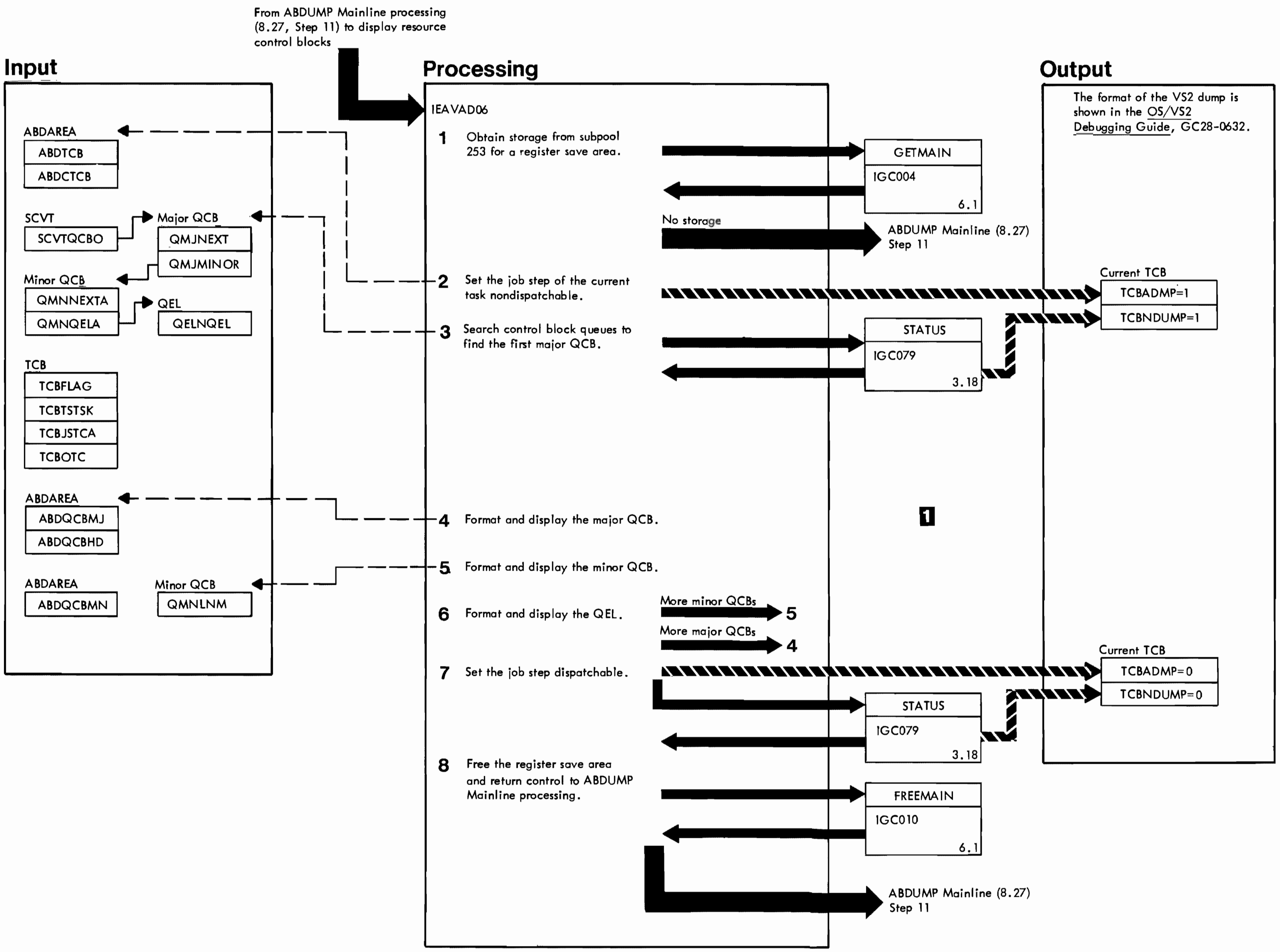
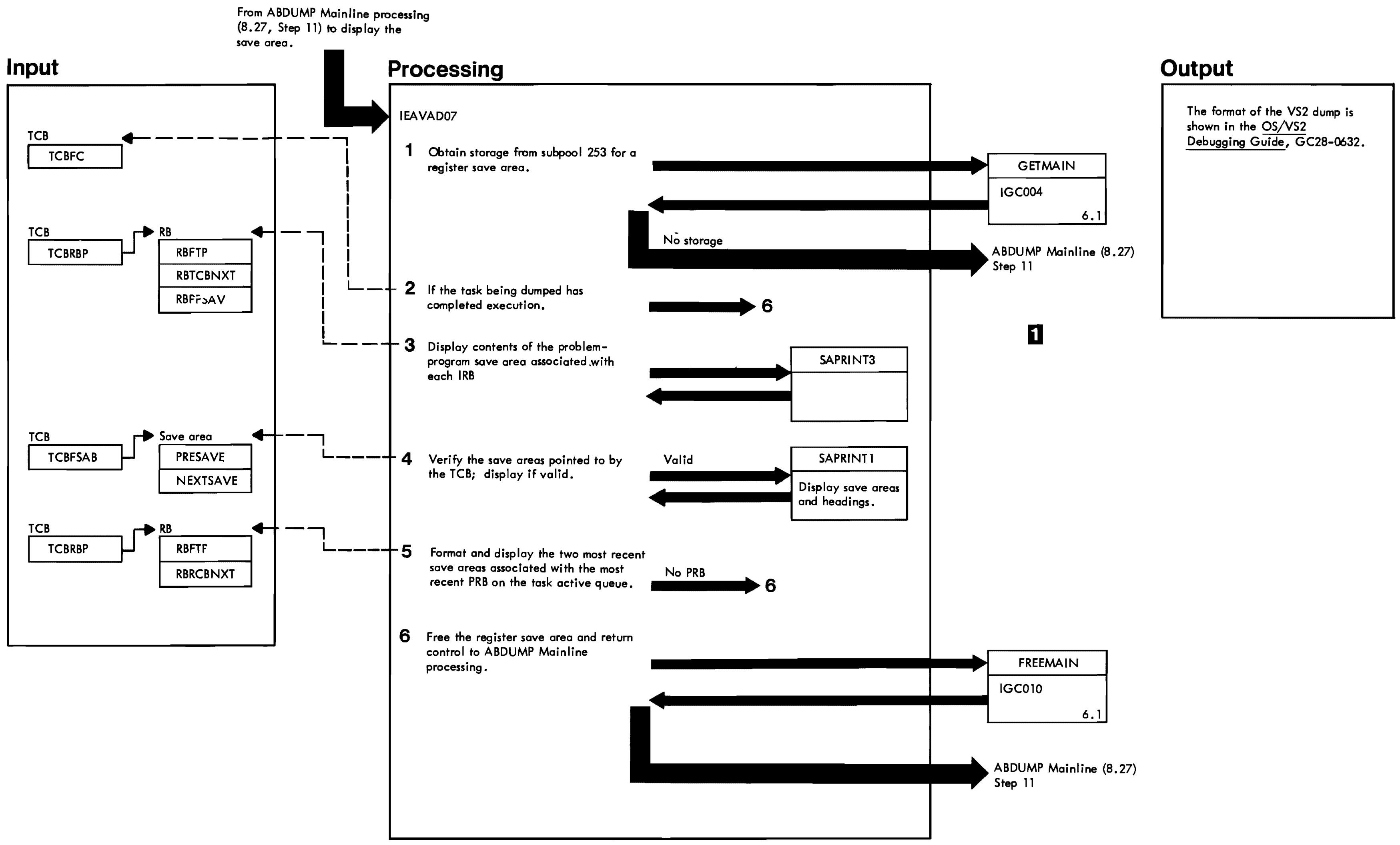

From ABDUMP Mainline processing (8.27, Step 11) to display resource control blocks
Input
ABDAREA
ABDTCB
ABDCTCB
SCVT
SCVTQCBO
Major QCB
QMJNEXT
QMJMINOR
Minor QCB
QMNNEXTA
QMNQELA
QEL
QELNQEL
TCB
TCBFLAG
TCBTSTSK
TCBJSTCA
TCBOTC
ABDAREA
ABDQCBMJ
ABDQCBHD
ABDAREA
ABDQCBMN
Minor QCB
QMNLNM
Processing
IEAVAD06
1 Obtain storage from subpool 253 for a register save area.
GETMAIN
IGC004
6.1
No storage
ABDUMP Mainline (8.27) Step 11
2 Set the job step of the current task nondispatchable.
3 Search control block queues to find the first major QCB.
STATUS
IGC079
3.18
4 Format and display the major QCB.
5 Format and display the minor QCB.
6 Format and display the QEL.
More minor QCBs
5
More major QCBs
4
7 Set the job step dispatchable.
STATUS
IGC079
3.18
8 Free the register save area and return control to ABDUMP Mainline processing.
FREEMAIN
IGC010
6.1
ABDUMP Mainline (8.27) Step 11
1
Output
The format of the VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.
Current TCB
TCBADMP=1
TCBNDUMP=1
Current TCB
TCBADMP=0
TCBNDUMP=0

Diagram 8.31 ABDUMP -- Formatting QCBs (Module IEAVAD06)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD06	
2 If the current task is being dumped (ABDTCB), the job step is set nondispatchable.		
3 The QEL (queue element) is found by searching the major and minor QCB (queue control block) chains. Once the QEL has been found, it is tested for the following conditions: • The task that requested the QEL is in the same job step as the task being dumped (TCBJSTCA). • The task is not a key 0 (TCBFLAG) nor a TSO task (TCBTSTSK), and the requesting task is the initiator of the job step (TCBOTC). If these conditions exist, the control blocks associated with this QEL can be displayed and formatted. If they do not, the next QEL is selected. Note: The search of the QCB queue must be performed disabled. Displaying the control block occurs in enabled state.		MOREQEL
4 Because it is possible to have more than one QEL belonging to a single job step associated with a major QCB, the ABDQCBMJ flag is checked to determine whether the major QCB was previously displayed. If this flag is off, the major QCB is formatted and displayed together with the heading if this is the first major QCB (ABDQCBHD). After the major QCB has been processed, the minor QCB associated with it is selected. When the last major QCB has been processed, ABDUMP enters clean-up.		DISPLAY MOREQCB
5 If the minor QCB has not been displayed (ABDQCBMN = 0), it is formatted and displayed. The maximum length for a minor QCB name is 52 bytes; a name that exceeds this limit is truncated. After the minor QCB has been processed, all QELs associated with it are formatted and displayed. When the last minor QCB has been processed, ABDQCBMJ is set to 0 and the next major QCB is selected.		FORMMIN MOREMIN

NOTES	ROUTINE NAME	LABEL
6 All QELs associated with the minor QCB are formatted and displayed according to their position in the queue. When the last QEL has been processed, ABDQCBMN is set to 0 and the next minor QCB is selected.	IEAVAD06	MOREQEL
7		COMEXIT
8 A return code is passed in register 15: X'00' - successful completion X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine.		
1 The following ABDUMP print routines are called: • FORMAT (IEAVAD31) Unpacks and translates data in the print line, providing an indentation factor. Called to execute Steps 4, 5, and 6. • FORMAT01 (IEAVAD41) Unpacks and translates data in the print line, without providing an indentation factor. Called to execute Steps 4 and 5. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute Steps 4, 5, and 6. Should any of the ABDUMP print routines fail (return code ≠ X'00'), control passes to cleanup processing (Step 7). Note: The appropriate label precedes each field.		

Diagram 8.32
ABDUMP—
Displaying the Save Area



1 Obtain storage from subpool 253 for a register save area.

2 If the task being dumped has completed execution.

3 Display contents of the problem-program save area associated with each IRB

4 Verify the save areas pointed to by the TCB; display if valid.

5 Format and display the two most recent save areas associated with the most recent PRB on the task active queue.

6 Free the register save area and return control to ABDUMP Mainline processing.

The format of the VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.32 ABDUMP -- Displaying the Save Area (Module IEAVAD07)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD07	
2 If the task has completed execution, there are no save areas to be displayed. The label for the trace is displayed however.		
3 If an invalid missing page interruption occurs, the display is terminated and the search continues for the next IRB. However, if the next RB on the queue (RBTCBNXT) is the oldest, processing continues with Step 4.		TESTIRB NEXTRB
4 Each problem program save area, beginning with the first (TCBFSAB), is tested by the subroutine VALID. Under the following conditions, the save area is considered valid: • The entire save area is addressable (this check is made by the Validity Check routine IEAOVI00). • The save area's forward pointer (NEXTSAVE) is not equal to the back pointer (PREVSAVE). • The back pointer is equal to the previous save area address. If condition 1 is not fulfilled, the display is terminated. If condition 2 or 3 is not fulfilled, the save area is displayed (using SAPRINT2) without headings.		FORTRC FORLOOP BADBACK
5 After the most recent RB has been selected, it is tested to determine whether it is a PRB (RBFTP = 0). If not, the next RB is selected and, when the end of the queue has been reached, exit processing is entered. Although a PRB exists, the save areas are not displayed under the following conditions: • The dump is not of the current task (ABDCTCB). • Save areas are not addressable. Instead, return is made to the caller with a return code of X'00'.		PRECODE RBLOOP PRBLOOP

NOTES	ROUTINE NAME	LABEL
6 If entry to this processing is a result of task completion (Step 2) or a missing PRB (Step 5), a code of X'00' is returned to ABDUMP Mainline processing. The return code is also X'00' if processing has completed successfully. It is X'04' if there was insufficient storage for the dump, or a failure occurred in an ABDUMP print routine.	IEAVAD07	AWAY
1 The following ABDUMP print routines are called: • FORMAT (IEAVAD31) Unpacks and translates data in the print line, providing an indentation factor. Called to execute Steps 3, 4, and 5. • FORMAT01 (IEAVAD41) Unpacks and translates data in the print line, without providing an indentation factor. Called to execute Steps 3, 4, and 5. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute Steps 3, 4, and 5. Should any of the ABDUMP print routines fail (return code ≠ X'00'), control passes to the cleanup processing (Step 6). _Note:_ The appropriate label precedes each field.		

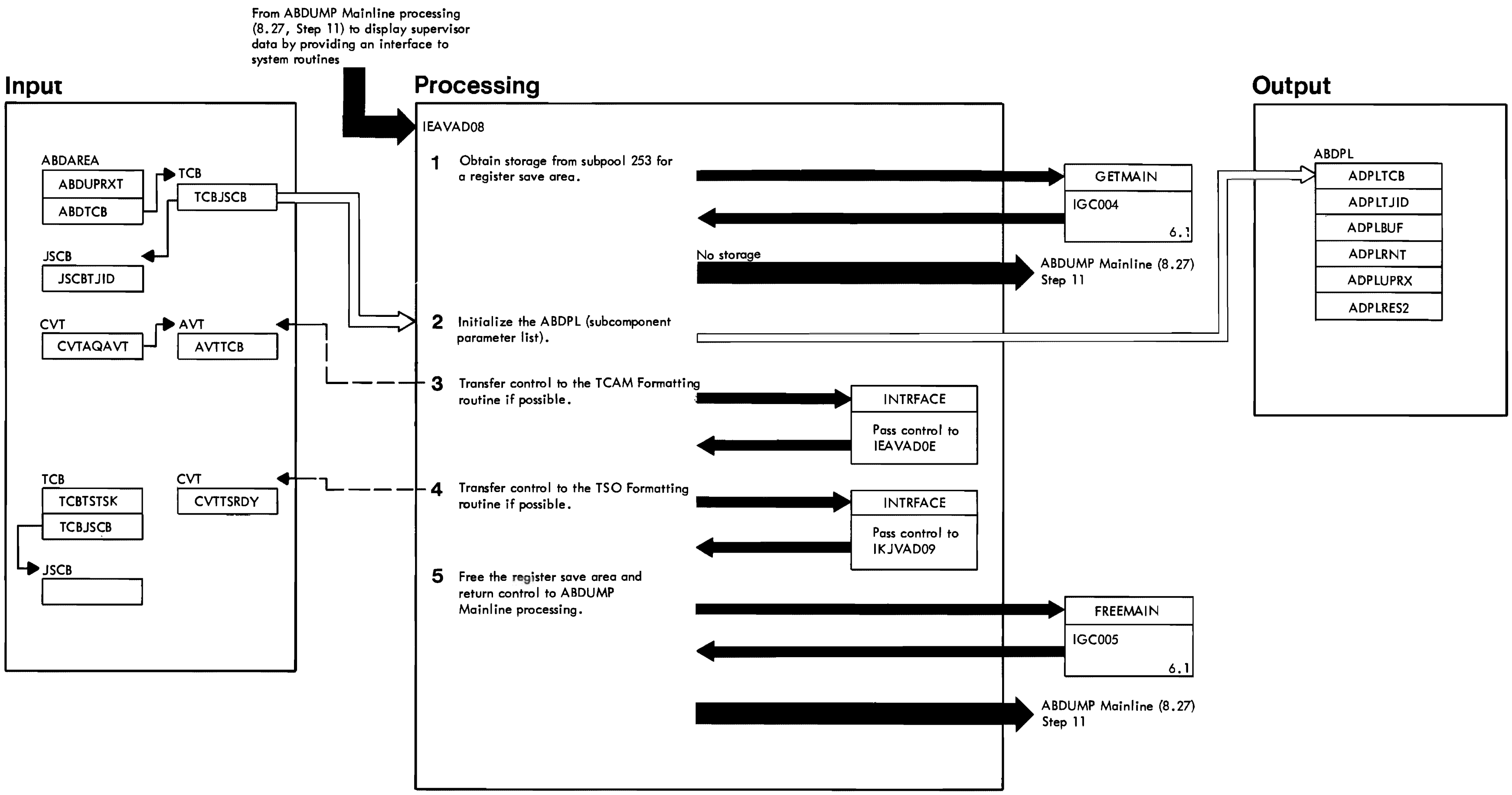

From ABDUMP Mainline processing (8.27, Step 11) to display supervisor data by providing an interface to system routines
Input
ABDAREA
ABDUPRXT
ABDTCB
TCB
TCBJSCB
JSCB
JSCBTJID
CVT
CVTAQAVT
AVT
AVTTCB
TCB
TCBTSTSK
TCBJSCB
CVT
CVTTSRDY
JSCB
Processing
IEAVAD08
1 Obtain storage from subpool 253 for a register save area.
GETMAIN
IGC004
6.1
No storage
ABDUMP Mainline (8.27) Step 11
2 Initialize the ABDPL (subcomponent parameter list).
3 Transfer control to the TCAM Formatting routine if possible.
INTRFACE
Pass control to IEAVAD0E
4 Transfer control to the TSO Formatting routine if possible.
INTRFACE
Pass control to IKJVAD09
5 Free the register save area and return control to ABDUMP Mainline processing.
FREEMAIN
IGC005
6.1
ABDUMP Mainline (8.27) Step 11
Output
ABDPL
ADPLTCB
ADPLTJID
ADPLBUF
ADPLRNT
ADPLUPRX
ADPLRES2

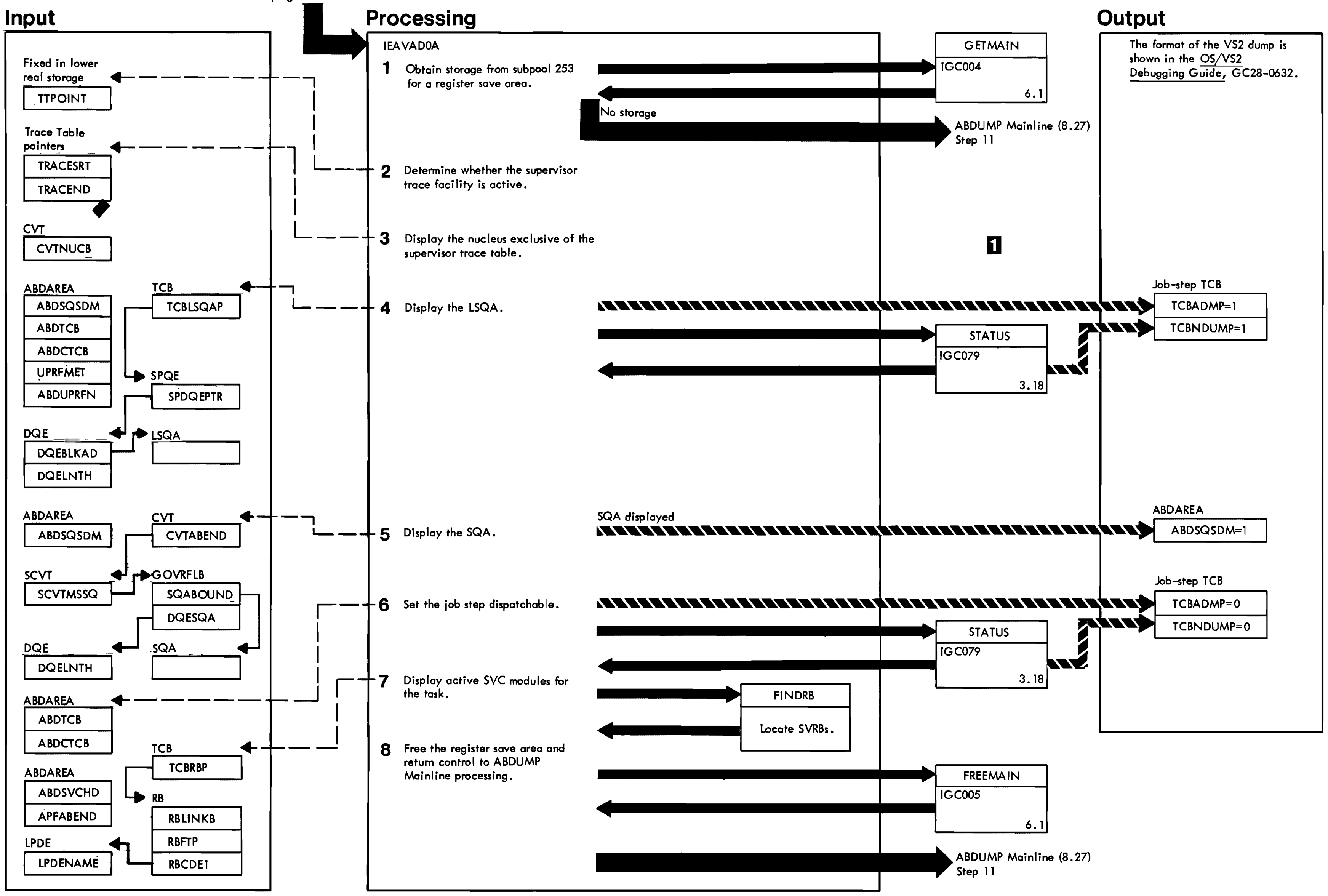
Diagram 8.33 ABDUMP Interface Routine (Module IEAVAD08)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD08	
2 The subcomponent parameter list is initialized because at least one formatting routine will get control. The parameter list is initialized to the following values: ADPLTCB = Address of TCB of task being dumped. ADPLTJID = Terminal job identifier (JSCBTJID), if there is a job-step control block. ADPLBUF = Address of print buffer. ADPLRNT = Address of PRINT routine (IEAVAD81). ADPLUPRX = Address of the exit routine ABDUPRXT. The processing of either the TCAM or TSO system formatting routine may cause an invalid missing page interruption. If so, the system routine can avoid abnormal termination of the task, and suspension of the dump, by placing the address of an exit routine in ABDUPRXT. If an invalid missing page interruption occurs, control returns to the system routine at the specified exit routine address. The exit routine restores registers 14-2 from the PIE and continues with exception handling. If an invalid missing page interruption does not occur, the system routine must set the exit routine address to 0. If an invalid missing page interruption occurs, and the exit routine address is 0, the task is abnormally terminated with a completion code of X'0C4'. ADPLRES2 = 4 words used by system formatting routines.		

NOTES	ROUTINE NAME	LABEL
3 The TCAM Formatting routine gains control (using the internal subroutine INTRFACE) if a TCAM AVT (address vector table) exists, and if the TCB being displayed is the TCB for the TCAM Message Control Program. The subroutine INTRFACE is given the entry-point name and an error message ID. The error message ID is necessary in case the formatting routine cannot allocate storage for work and save areas. INTRFACE then loads the requested module, passes control to it, and deletes it upon completion of processing. If INTRFACE receives a return code other than 0, it prints an error message and returns control to the caller. Otherwise it returns directly to the caller. A description of the TCAM Formatting routine is available in <u>OS/VS TCAM Logic</u>.	IEAVAD08	INTRFACE
4 The TSO Formatting routine gains control if the following are true: • The task being dumped is a TSO task (TCBTSTSK = 1). • Time sharing is active (CVTTSRDY = 1). • A JSCB exists (contents of register 5 ≠ 0). The subroutine INTRFACE (see Step 3) provides the necessary linkage. A description of the TSO Formatting routine is available in <u>OS/VS2 TSO Control Program Logic</u>.		INTRFACE
5 A return code is passed in register 15: X'00' - successful completion. X'04' - insufficient storage for the dump.		FREERET

Diagram 8.34 ABDUMP— Displaying the Nucleus

From ABDUMP Mainline processing (8.27, Step 11) to display the control program nucleus



1 Obtain storage from subpool 253 for a register save area.

2 Determine whether the supervisor trace facility is active.

3 Display the nucleus exclusive of the supervisor trace table.

4 Display the LSQA.

5 Display the SQA.

6 Set the job step dispatchable.

7 Display active SVC modules for the task.

8 Free the register save area and return control to ABDUMP Mainline processing.

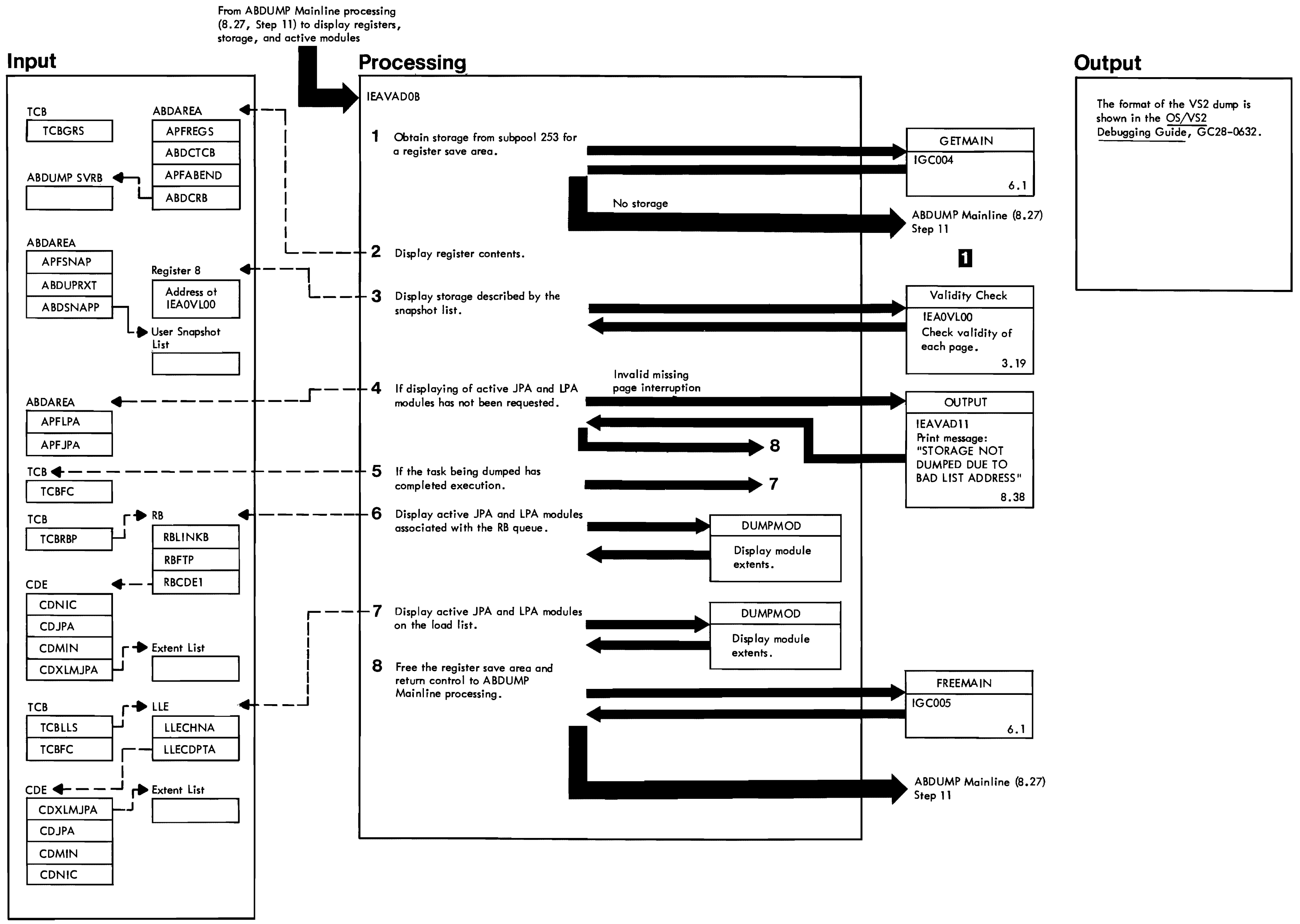
The format of the VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.34 ABDUMP -- Displaying the Nucleus (Module IEAVAD0A)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD0A	
2 The nucleus is displayed in two sections if part of the nucleus is above the trace table. The trace table is often the last block in the nucleus.		
3 If the supervisor trace is active, the nucleus is first displayed up to the start of the trace table (TRACESRT). The end of the trace table is then found (TRACEND+32). If there is more nucleus to display, a continuation message is printed and the second part of the nucleus is displayed.		
4 If the current task is being dumped, the job step is set nondispatchable to prevent changes in the LSQA when the dump is being taken. If there is no SPQE (subpool queue element) for the LSQA, execution continues with Step 5. The length and address are found from the DQE and FORMET (IEAVAD71) is called to		LSQA
print the LSQA. If an invalid missing page interruption occurs during IEAVAD71 processing, a skip is made to the next page where the display continues.		LSQALOOP
5 If the SQA has not been displayed (ABDSQSDM = 0), it is processed in a manner similar to the LSQA. Invalid missing page interruptions are again expected when IEAVAD71 is called.		SQA
6 If the current task is being dumped, the job step is reset dispatchable.		SVCMODS
7 If the task being dumped is not the current task, processing begins with the RB chained to the TCB of the task being dumped. The SVRB following the ABDUMP SVRB is obtained; if ABEND called ABDUMP (APFABEND = 0), the ABEND SVRB is bypassed.		

NOTES	ROUTINE NAME	LABEL
The FINDRB subroutine is called twice; the first time to scan the RB queue for SVRBs and the second time to search for an SVRB with a name equal to the name found in the initial search. If FINDRB has been called to locate SVRBs, the contents supervision routine IEAQCDSR is called to locate the CDE for the module on the link pack area queue. If the CDE is not found, IEAVVMSR is called to obtain the LPDE (link pack directory entry) for the SVC name. If the LPDE is found, the name of the module (found in LPDENAME) is returned to the caller along with the SVRB address. FINDRB is again called to search for an SVRB of a module with the same name as that in LPDENAME. The address of the two SVRBs are compared, and, if equal, the module is displayed. If a match is not found, the module was displayed earlier when the previous SVRB was processed. After the display of the module has been completed, the search is renewed for a new SVRB. When all modules have been displayed, clean-up processing is entered. The following types of SVRBs are processed by the FINDRB subroutine: • SVRB for a type 3-SVC or the first load of a type-4 SVC. • SVRB for a non-first load of a type-4 paged SVC. • SVRB for a non-first load of a fixed SVC.	IEAVAD0A	NEXTRB
8 A return code is passed in register 15: X'00' - successful completion X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine.		FREERET
1 The following ABDUMP print routines are called: • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute steps 1, 3, 4, 5, and 7. • FORMET (IEAVAD71) Prints blocks of storage. Called to execute steps 3, 4, 5, and 7. Should either ABDUMP print routine fail (return code ≠ 0), the job step is set dispatchable prior to passing control to cleanup processing (Step 8). Note: The appropriate label precedes each field.		

Diagram 8.35 ABDUMP— Displaying Registers


From ABDUMP Mainline processing (8.27, Step 11) to display registers, storage, and active modules

IEAVAD0B

1 Obtain storage from subpool 253 for a register save area.

2 Display register contents.

3 Display storage described by the snapshot list.

4 If displaying of active JPA and LPA modules has not been requested.

5 If the task being dumped has completed execution.

6 Display active JPA and LPA modules associated with the RB queue.

7 Display active JPA and LPA modules on the load list.

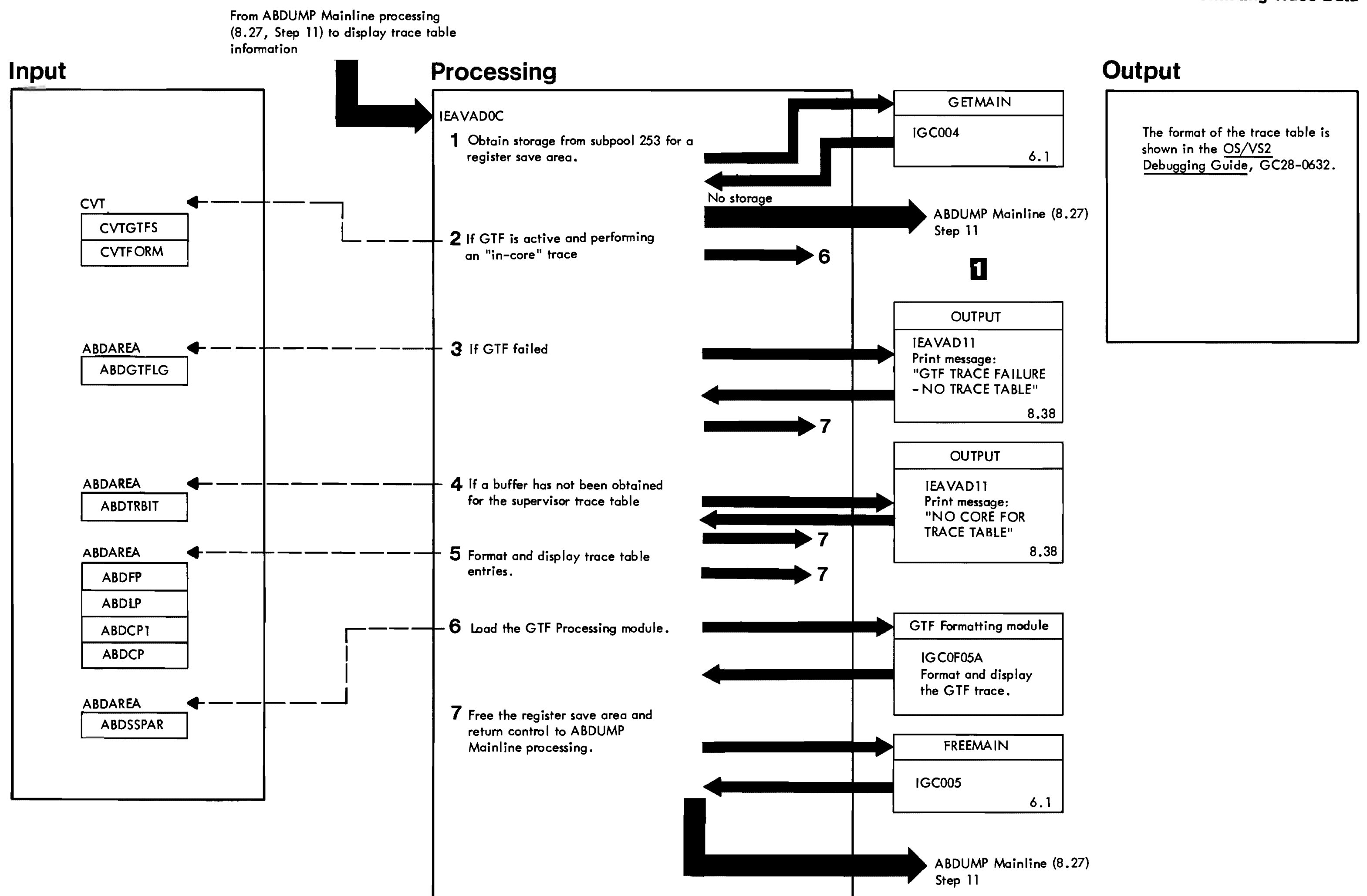
8 Free the register save area and return control to ABDUMP Mainline processing.

The format of the VS2 dump is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.35 ABDUMP -- Displaying Registers (Module IEAVAD08)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD0B	
2 The register contents are displayed if requested (APFREGS = 1). If the task being dumped is the current task, the floating-point registers are displayed; they are stored in the current TCB during the formatting and displaying. The general registers are found in the ABDUMP SVRB (SNAP called ABDUMP) or the ABEND SVRB (ABEND called ABDUMP). If the current task is not being dumped, the floating-point and general registers are found in the TCB of the task being dumped.		
3 The snapshot list is displayed if supplied and requested by the user (APFSNAP = 1). If an invalid missing page interruption occurs while fetching an address or length from the list, an error message is displayed and execution continues with Step 3. The Validity Check routine (IEA0VL00) is called to verify each page containing data to be displayed. If a page is invalid, the list entry is ignored and the next entry is processed. Otherwise, the storage described by the snapshot list entry is displayed.		UPIT FAILURE VALIDLST NEXTONE GOOD
4		PASSUPR
6 If the task has not completed execution, all modules associated with RBs on the active queue are displayed if the following conditions exist: • The RB is a PRB (RBFTP = 0). • A CDE exists (RBCDE1 ≠ 0). • The module is "in-core" (CDNIC = 0). The RB queue is searched beginning with the oldest RB. After the RB queue has been exhausted, the load list is processed (Step 7). The internal subroutine DUMPMOD is passed the address of a CDE. After verifying that the module's queue is to be displayed, DUMPMOD locates extent lists for the module and calls FORMET (IEAVAD71) to print these extents.		RBLOOP1 DUMPMOD DOIT
7 For each LLE (load list element) on the active task queue, all associated modules are displayed if the following conditions exist: • The module is "in-core" (CDNIC = 0). • The CDE is not queued to an RB; if it is, the RB must not be on the active queue of the task being dumped. The internal subroutine DUMPMOD is again called to process the extents for the module.	IEAVAD0B	LLELOOP
8 A return code is passed in register 15: X'00' - successful completion. X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine.		FREERET
1 The following ABDUMP print routines are called: • FORMAT22 (IEAVAD61) Unpacks and translates data in the print line. Called to execute Steps 2, 6, and 7. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute Steps 2, 3, 6, and 7. • FORMET (IEAVAD71) Prints blocks of storage. Called to execute Steps 3, 6, and 7. Should any of the ABDUMP print routines fail (return code ≠ X'00'), control passes to cleanup processing (Step 8). Note: The appropriate label precedes each field.		

Diagram 8.36
ABDUMP—
Formating Trace Data



1 Obtain storage from subpool 253 for a register save area.

2 If GTF is active and performing an "in-core" trace

3 If GTF failed

4 If a buffer has not been obtained for the supervisor trace table

5 Format and display trace table entries.

6 Load the GTF Processing module.

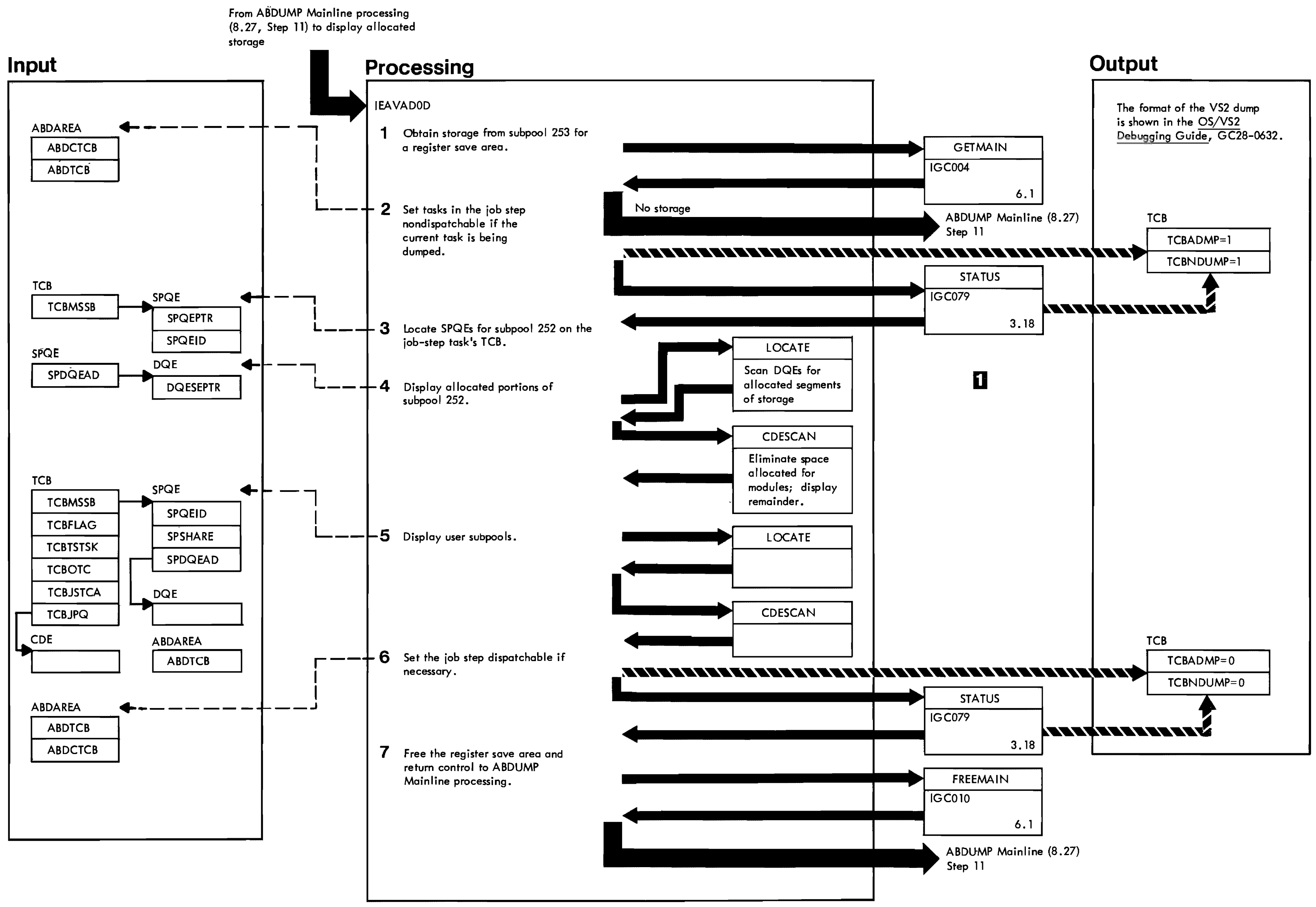
7 Free the register save area and return control to ABDUMP Mainline processing.

The format of the trace table is shown in the OS/VS2 Debugging Guide, GC28-0632.

Diagram 8.36 ABDUMP -- Formatting Trace Data (Module IEAVAD0C)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVAD0C	
3 If ABDGTFLG is on, ABDUMP issued a HOOK macro instruction to suspend GTF. The suspension of the trace indicates there was a failure in GTF. After printing the error message, the return code is saved and control passes to cleanup processing (Step 7).		OUTLINE
4 Upon return from OUTPUT, the return code is saved and control passes to cleanup processing (Step 7).		
5 The trace table entries are formatted and displayed beginning with the oldest entry (the one following the new current entry ABDCP). All entries existing between the old current (ABDCP1) and the new current (ABDCP), are invalid because they occurred during the time the trace table was moved (IEAVAD01 execution). If GTF suspended the supervisor trace, a flag (SUSPEND) is set in the current trace entry. Before each entry is formatted, this flag is tested. If it is set, OUTPUT is called to print a message indicating that there may be trace entries that are not displayed in the table. After all entries have been displayed, the trace table buffer is freed.		CHKEND FREEBUF
Note: The trace table is provided only in a SNAP dump.		
6 The interface with IGC0F05A is established using ABDSSPAR as the parameter list. On return from IGC0F05A, if the return code is not 0, a return code of X'10' is passed to ABDUMP Mainline processing.		
7 A return code is passed in register 15: X'00' - successful completion. X'04' - an ABDUMP print routine failed while executing. X'0C' - storage was not obtained for the register save area. ABDUMP Mainline processing must free the trace table buffer if it was obtained earlier.		FREERET
1 The following ABDUMP print routines are called: • FORMAT (IEAVAD31) Unpacks and translates data in the print line, providing an indentation factor. Called to execute Step 5. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute steps 3, 4, and 5. Should either ABDUMP print routine fail (return code ≠ X'00'), control passes to cleanup processing (Step 7). _Note:_ The appropriate label precedes each field.		

Diagram 8.37 ABDUMP— Displaying Subpools



1 Obtain storage from subpool 253 for a register save area.

2 Set tasks in the job step nondispatchable if the current task is being dumped.

3 Locate SPQEs for subpool 252 on the job-step task's TCB.

4 Display allocated portions of subpool 252.

5 Display user subpools.

6 Set the job step dispatchable if necessary.

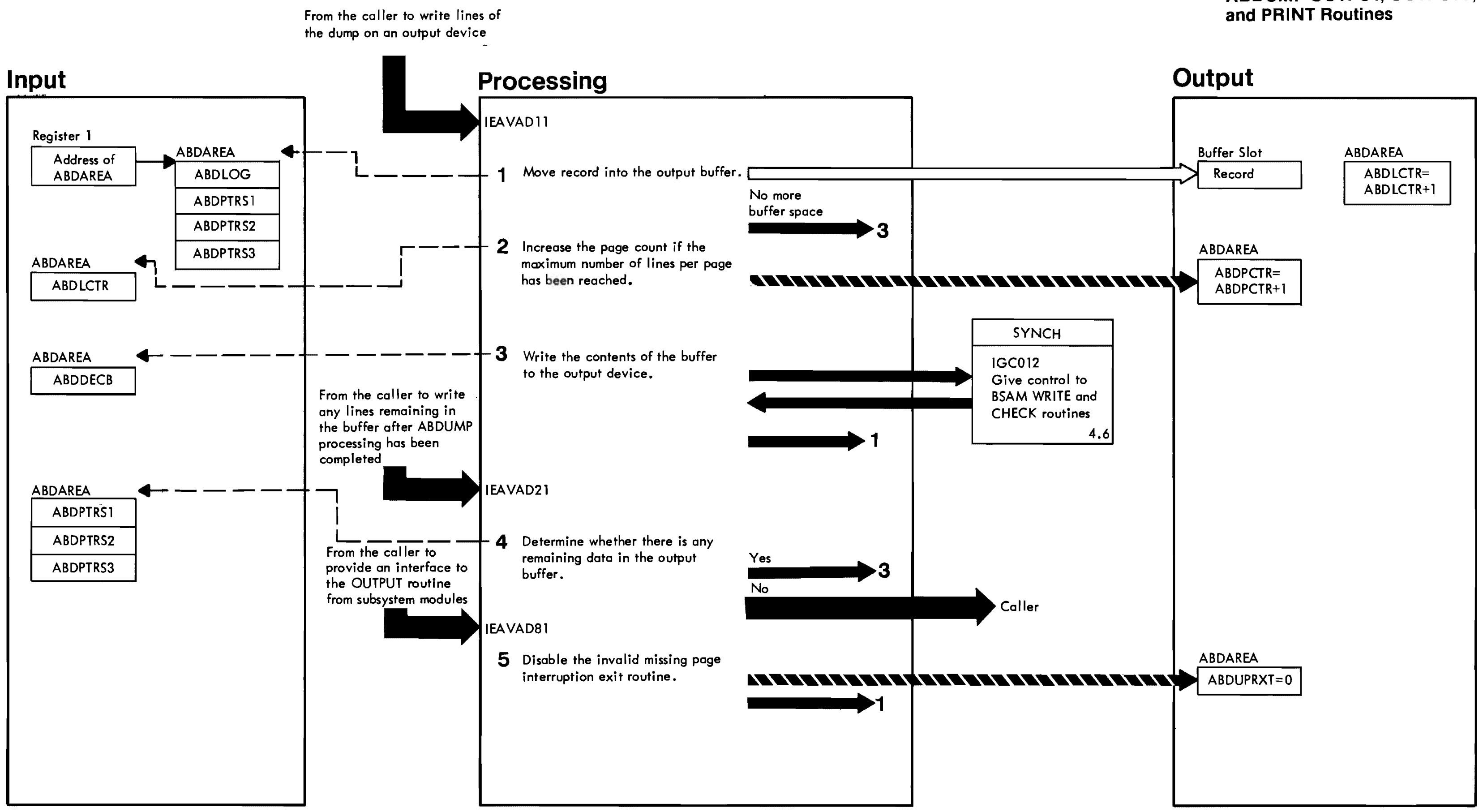
7 Free the register save area and return control to ABDUMP Mainline processing.

Diagram 8.37 ABDUMP -- Displaying Subpools (Module IEAVADOD)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed if storage is not available.	IEAVADOD	
2 The job step must be set nondispatchable to ensure that storage contents are accurate. If the current task is not being dumped, the job step is already nondispatchable.		
3 If the SPQE is not for subpool 252, the next SPQE is obtained. When the last SPQE has been searched (SPQEPTR = 0) and subpool 252 does not exist, execution continues with Step 5.		SP252RT
4 The LOCATE subroutine is called with the address of the first DQE for a subpool that is to be displayed. For each DQE on the queue, allocated segments are isolated one at a time. The CDESCAN subroutine is then called to process these segments by eliminating all space allocated for modules and displaying the remainder.	LOCATE LOCATE CDESCAN CDESCAN	MAKECK GOSCAN REPSCAN TRYPRINT
5 Displaying is performed for all the subpools described by SPQEs on the task queue of the task being dumped under the following conditions:	IEAVADOD	MAIN
• The subpool is a user subpool (0-127).		MAINLOOP
• The task being dumped owns the subpool (SPSHARE = 0)		
• The dumped task is not the owner and (1) has a key other than 0 (TCBFLAG ≠ 0), and (2) is not a TSO task (TCBTSTSK ≠ 1).		NOTOWNER
• The task is not the owner and either has a key of 0 or is a TSO task. The task owning the subpool is in the same job step (TCBJSTCA) as the dumped task.		CKNXTSPQ

NOTES	ROUTINE NAME	LABEL
After a valid SPQE has been found, the DQE for this subpool is found and the LOCATE routine is called. The processing performed by the LOCATE and CDESCAN routines is explained in Step 4.	IEAVADOD	GETDQE FROMSHAR
Upon successful return from the LOCATE routine, the next SPQE for a user subpool is processed.		MAINLOOP
6		EXITA
7 A return code is passed in register 15: X'00' - successful completion X'04' - insufficient storage for the dump, or a failure in an ABDUMP print routine.		FREE
1 The following ABDUMP print routines are called: • FORMET (IEAVAD71) Prints blocks of storage. Called to execute Steps 4 and 5. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to print any messages that might be needed. Should either of the ABDUMP print routines fail (return code ≠ X'00'), control passes to cleanup processing (Step 6). Note: The appropriate label precedes each field.		

Diagram 8.38
ABDUMP OUTPUT, OUTPUT5, and PRINT Routines



1 Move record into the output buffer.

2 Increase the page count if the maximum number of lines per page has been reached.

3 Write the contents of the buffer to the output device.

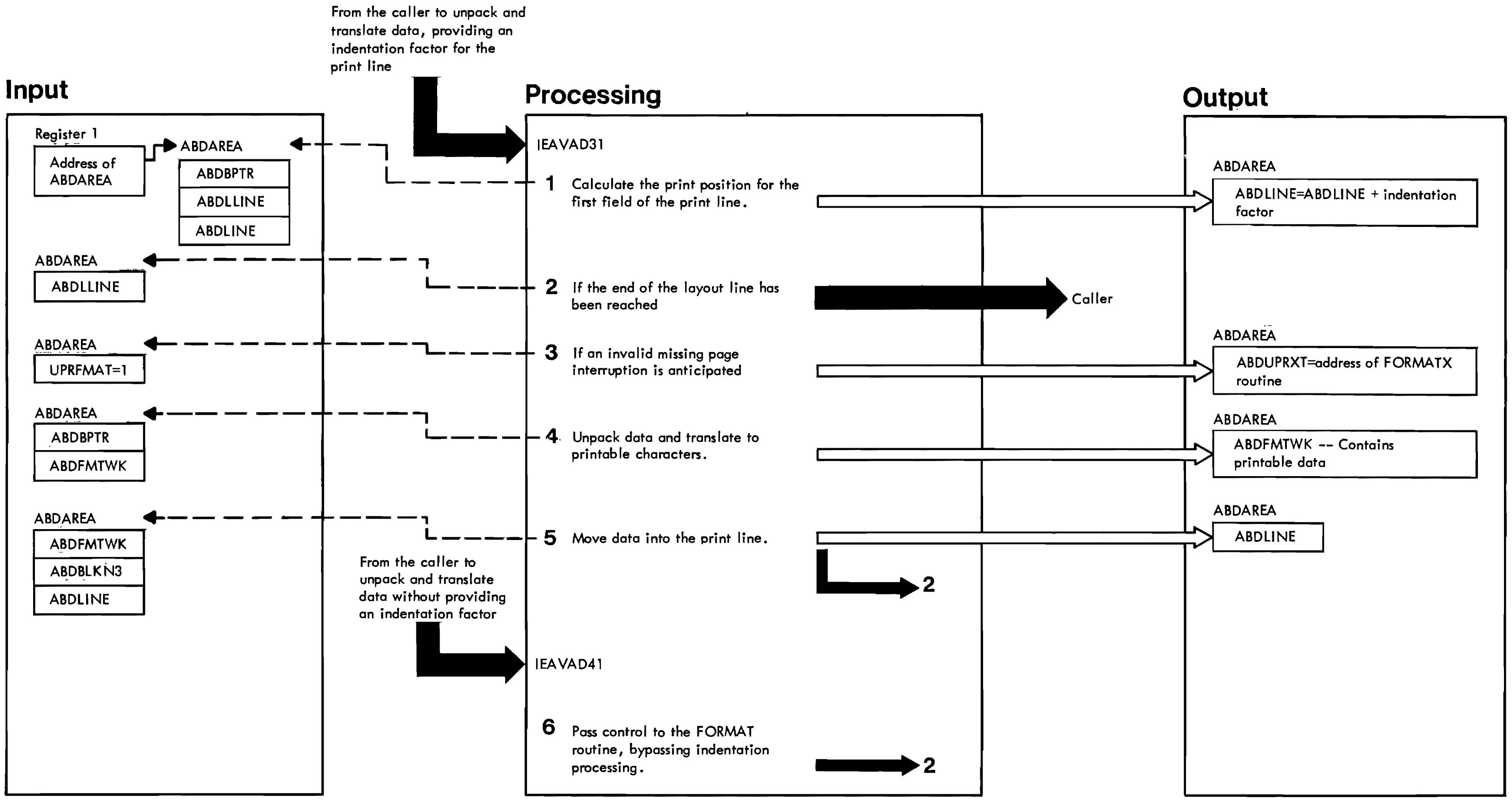
4 Determine whether there is any remaining data in the output buffer.

5 Disable the invalid missing page interruption exit routine.

Diagram 8.38 ABDUMP OUTPUT, OUTPUT5, and PRINT Routines (Module IEAVAD11)

NOTES	ROUTINE NAME	LABEL
1 The output record, starting from ABDLOG, is moved to the output buffer for a length of 125 bytes (a 4-byte record descriptor word and 121 bytes of data). The line count is increased by 1. If there is insufficient space (125 bytes) in the buffer for another record, preparations are made to write the contents of the buffer (Step 3).	OUTPUT	RECLOOP
2 The maximum number of lines per page is 56. When this number has been reached, the page count is increased and a carriage control character is set to eject to a new page.		PAGEFULL
3 The SYNCH macro instruction is used to enter the internal subroutine WRITCHEK, which issues WRITE and CHECK macro instructions. The SYNCH macro instruction is used to ensure that the WRITCHEK subroutine is entered with the user protection key.		BUFFULL
4	OUTPUT5	
5 Before disabling the exit routine, register 1 is loaded with the address of ABDAREA, which is located in the second word of the ESA. This routine is called by the subsystem routines TCAM and GTF to provide the correct interface to the OUTPUT routine (IEAVAD11). Exits to handle invalid missing page interruptions are prohibited to prevent careless use of the field by the subsystem modules.	PRINT	

Diagram 8.39
ABDUMP FORMAT and FORMAT01 Routines

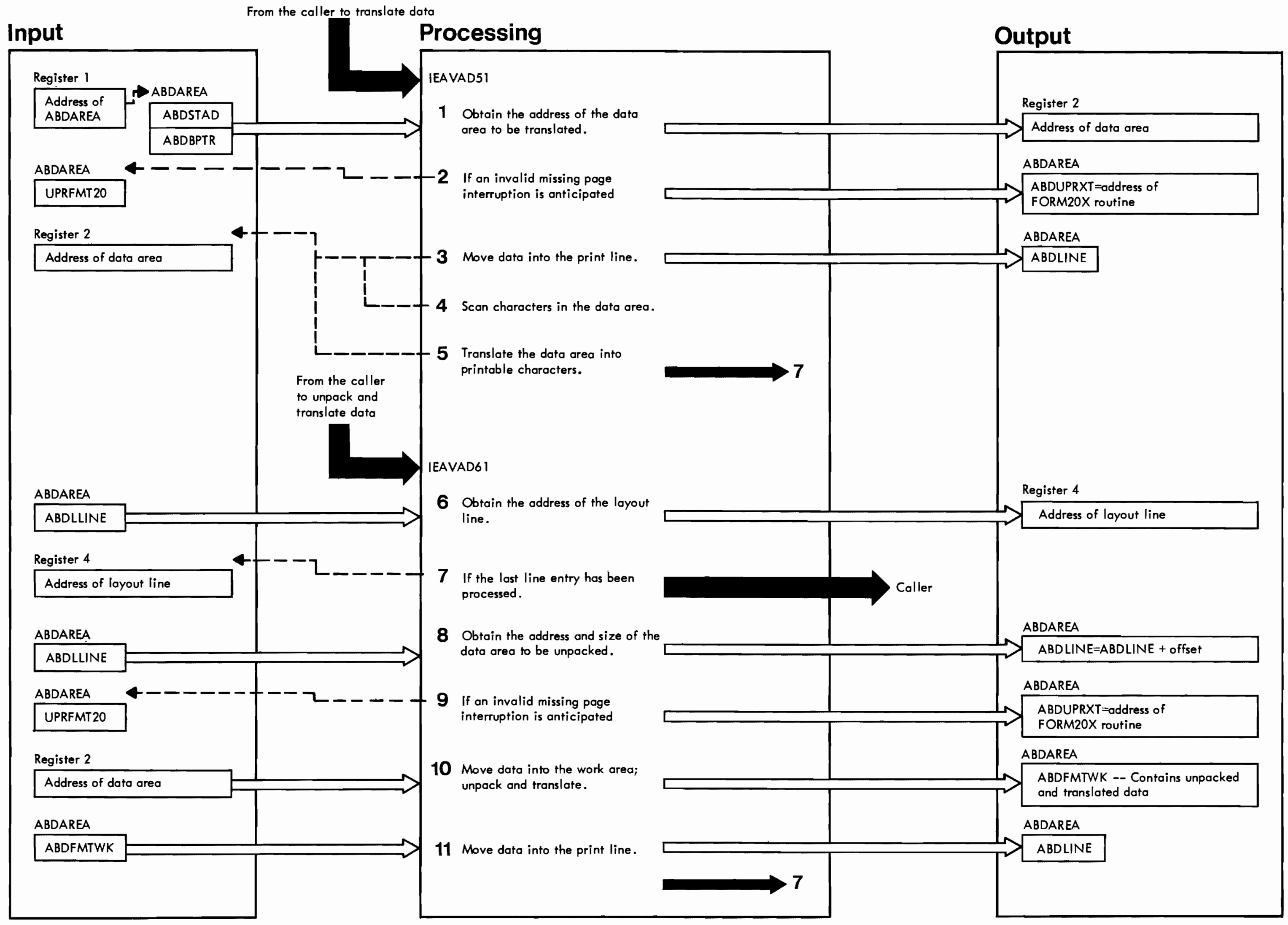


From the caller to unpack and translate data, providing an indentation factor for the print line
Input
Register 1
Address of ABDAREA
ABDAREA
ABDBPTR
ABDLLINE
ABDLINE
ABDAREA
ABDLLINE
ABDAREA
UPRFMAT=1
ABDAREA
ABDBPTR
ABDFMTWK
ABDAREA
ABDFMTWK
ABDBLKN3
ABDLINE
Processing
IEAVAD31
1 Calculate the print position for the first field of the print line.
2 If the end of the layout line has been reached
Caller
3 If an invalid missing page interruption is anticipated
4 Unpack data and translate to printable characters.
5 Move data into the print line.
2
From the caller to unpack and translate data without providing an indentation factor
IEAVAD41
6 Pass control to the FORMAT routine, bypassing indentation processing.
2
Output
ABDAREA
ABDLINE=ABDLINE + indentation factor
ABDAREA
ABDUPRXT=address of FORMATX routine
ABDAREA
ABDFMTWK -- Contains printable data
ABDAREA
ABDLINE

Diagram 8.39 ABDUMP FORMAT and FORMAT01 Routines (Module IEAVAD31)

NOTES	ROUTINE NAME	LABEL
1 The layout line (ABDLLINE) contains the label of the field immediately followed by the value of the field. The indentation factor (the first byte of the layout line) determines how many spaces from the beginning of the print line (ABDLINE) a particular field starts. The print position for the first field of the print line is obtained by adding the indentation factor specified in the layout line to the beginning of the print line. The layout line pointer is then increased past the indentation factor. The address of the control block to be formatted is found in ABDBPTR.	FORMAT	OLDVAL
2 If the end of the layout line has not been reached (the delimiter character X'FF' has not been encountered), the label is obtained and moved into the print line. If control is returned to the caller, a code of X'00' is passed.		COMMON
3 If the data to be formatted results in an invalid missing page interruption, the address of the exit routine to handle these interruptions (FORMATX) is placed in ABDAREA. Should an invalid missing page interruption occur, FORMATX is called to restore registers 14-2 from the PIE. Return is then made to the caller with a return code of X'08'.	FORMATX	FORMATX
4 The data to be formatted is moved into a work area (ABDFMTWK) where it is unpacked and translated into EBCDIC characters.	FORMAT	NOUPR
5 After the data has been moved, the layout and print line counters are increased for the next entry. The number of blanks between fields in the print line is assumed to be 3 unless specified otherwise by the user.		BUMPLINE SPECSPAC
6 In bypassing indentation processing, the first print position is assumed to be correct.	FORMAT01	NEWVAL

Diagram 8.40
ABDUMP FORMAT20 and FORMAT22 Routines



IEAVAD51

1 Obtain the address of the data area to be translated.

2 If an invalid missing page interruption is anticipated

3 Move data into the print line.

4 Scan characters in the data area.

5 Translate the data area into printable characters.

IEAVAD61

6 Obtain the address of the layout line.

7 If the last line entry has been processed.

8 Obtain the address and size of the data area to be unpacked.

9 If an invalid missing page interruption is anticipated

10 Move data into the work area; unpack and translate.

11 Move data into the print line.

Diagram 8.40 ABDUMP FORMAT20 and FORMAT22 Routines (Module IEAVAD51)

NOTES	ROUTINE NAME	LABEL
1 FORMAT20 provides an EBCDIC translation of a line (32 bytes) of virtual storage. The data area begins at the address specified in ABDSTAD. The data is rounded to a 32-byte boundary and extends for 32 bytes. The location counter (ABDBPTR) is the virtual address of the data being dumped. The value of the location counter is expanded to six hexadecimal characters, positioned at the beginning of the print line (ABDLINE). Following the location counter are eight fullwords that are unpacked into two groups of four words each. This is the data at the address specified by the location counter.	FORMAT20	BASE1
2 If an invalid missing page interruption occurs, the FORM20X routine is entered to perform the following processing: • Restore registers 14-2 from the PIE. • Blank the print line (ABDLINE). • Return control to the caller with a return code of X'08'.	FORM20X	FORM20X
3 After moving the data, invalid missing page interruptions are disabled.	FORMAT20	NOUPR20
4 An asterisk precedes and follows the translated data. If the character is neither alphanumeric nor a blank, it is replaced by a character that translates into a period. All 32 characters are processed in this way.		LOOP GOLOOP
5 After translating the characters, FORMAT22 (IEAVAD61) is called with the layout line (register 4) set to the standard layout line for unpacking the location counter.		
6	FORMAT22	BASE2
7 If the layout line entry delimiter (X'FF') has been encountered, pointers are saved and control is returned to the caller.		REPEAT EXIT
8 The displacement is added to the print line to obtain the correct location in which to place the data.		
10		NOUPR22
11 After moving the data to the print line, the layout line counter is increased to the next entry and processing continues with Step 7.		

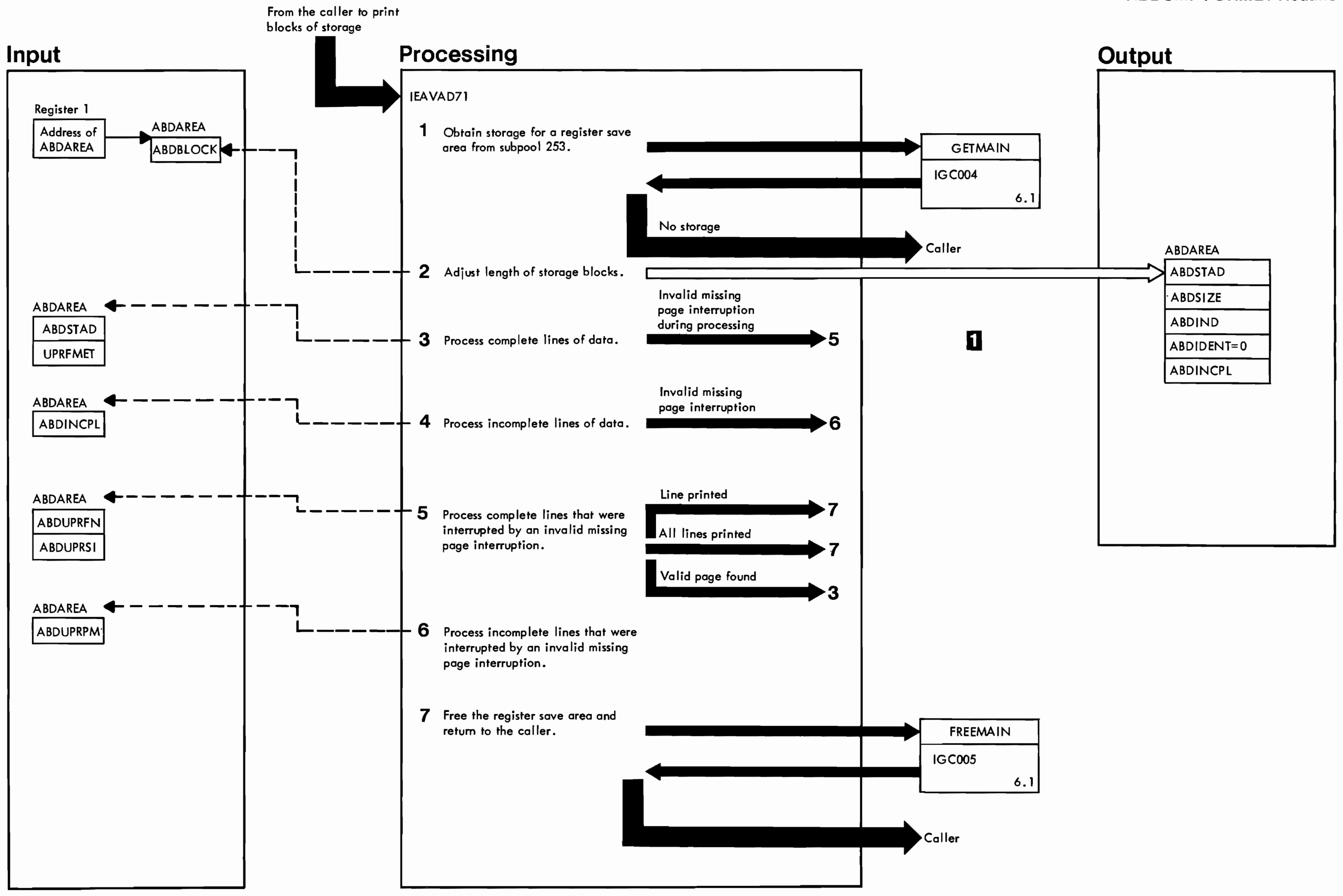

From the caller to print blocks of storage
Input
Register 1
Address of ABDAREA
ABDAREA
ABDBLOCK
ABDAREA
ABDSTAD
UPRFMET
ABDAREA
ABDINCPL
ABDAREA
ABDUPRFN
ABDUPRSI
ABDAREA
ABDUPRPM
Processing
IEAVAD71
1 Obtain storage for a register save area from subpool 253.
GETMAIN
IGC004
6.1
No storage
Caller
2 Adjust length of storage blocks.
3 Process complete lines of data.
Invalid missing page interruption during processing
5
4 Process incomplete lines of data.
Invalid missing page interruption
6
5 Process complete lines that were interrupted by an invalid missing page interruption.
Line printed
7
All lines printed
7
Valid page found
3
6 Process incomplete lines that were interrupted by an invalid missing page interruption.
7 Free the register save area and return to the caller.
FREEMAIN
IGC005
6.1
Caller
Output
ABDAREA
ABDSTAD
ABDSIZE
ABDIND
ABDIDENT=0
ABDINCPL
1

Diagram 8.41 ABDUMP FORMET Routine (Module IEAVAD71)

NOTES	ROUTINE NAME	LABEL
1 A return code of X'04' is passed to the caller.	FORMET	ENDID
2 The FORMET routine displays blocks of storage which are a multiple of fullwords. The starting address provided by the user (ABDSTAD) is rounded down to a fullword boundary; the number of bytes to be printed (ABDSIZE) is rounded up to a fullword boundary. The indentation factor and identical line counter (ABDIDENT) are set to 0. Because lines are printed on 32-byte boundaries, the starting address is rounded down to a multiple of 32 bytes. If an indentation factor is required, it is saved in ABDIND. The number of lines to be printed is then calculated. ABDINCPL contains the number of bytes in the last, incomplete line.		FORMET01
3 The starting address is rounded down to a 32-byte boundary. FORMAT20 is called to unpack the location counter address into printable hexadecimal, and translate the 32 bytes of data into EBCDIC. If invalid missing page interruptions are expected (UPRFMET=1), and one is encountered during FORMAT20 processing, control passes to Step 5.		COMPLINE TOVAD51
If an invalid missing page interruption has not occurred, FORMAT22 is invoked to unpack the data line. For the first complete line the indentation factor is calculated and the line is adjusted accordingly. Invalid missing page interruptions are processed as described above.		FORMET03
If FORMAT22 processing did not result in an invalid missing page interruption, the OUTPUT routine is invoked to print the data line. Upon return from OUTPUT, tests are made to determine whether the next line to be printed is identical to the previously printed line. If yes, an identical line message is printed as follows: LINE xxxxxx SAME AS ABOVE -- one identical line LINES xxxxxx - yyyyyy SAME AS ABOVE -- more than one identical line If the line is not identical, it is processed as described earlier.		FORMET04
After all complete lines have been processed, any incomplete line is printed (Step 4).		FORMET08

NOTES	ROUTINE NAME	LABEL
4 When a partial data line exists (less than 32 bytes), FORMAT20 is called to unpack the location counter address and translate the data line into EBCDIC. FORMAT22 is called to unpack the data line. When the incomplete line has been formatted, it is printed (using the OUTPUT routine) and control returns to the caller.	FORMET	INCPLINE FORMET09 FORMET10
5 If an invalid missing page interruption occurred during UPR FORMAT20 or FORMAT22 processing, several options are possible: • Control is immediately returned to the caller (Step 7).		
• The line that caused the invalid missing page interruption is skipped; the next valid defined page is found and processing continues from Step 3. If all lines have been printed, exit processing (Step 7) is entered.		SKIPONLY EXITBACK
• A message is printed describing the invalid lines: LOCATIONS xxxxxx - yyyyyy NOT DEFINED Processing for the next valid defined page continues as described above.		FINISH
6 If a message is requested, it also reads as follows: LOCATIONS xxxxxx - yyyyyy NOT DEFINED		LASTUPR
7 A return code of X'00' is provided for the caller.		NORMEXIT
1 The following ABDUMP print routines are called: • FORMAT20 (IEAVAD51) Translates data in the print line. Called to execute Steps 3 and 4. • OUTPUT (IEAVAD11) Prints lines of the dump on an output device. Called to execute Steps 3 and 4; also Steps 5 and 6 when a message is requested. • FORMAT22 (IEAVAD61) Unpacks and translates data in the print line. Called to execute Steps 3 and 4; also Steps 5 and 6 when a message is requested.		

Diagram 8.42
Overview of the STA Services
and ASIR Routines

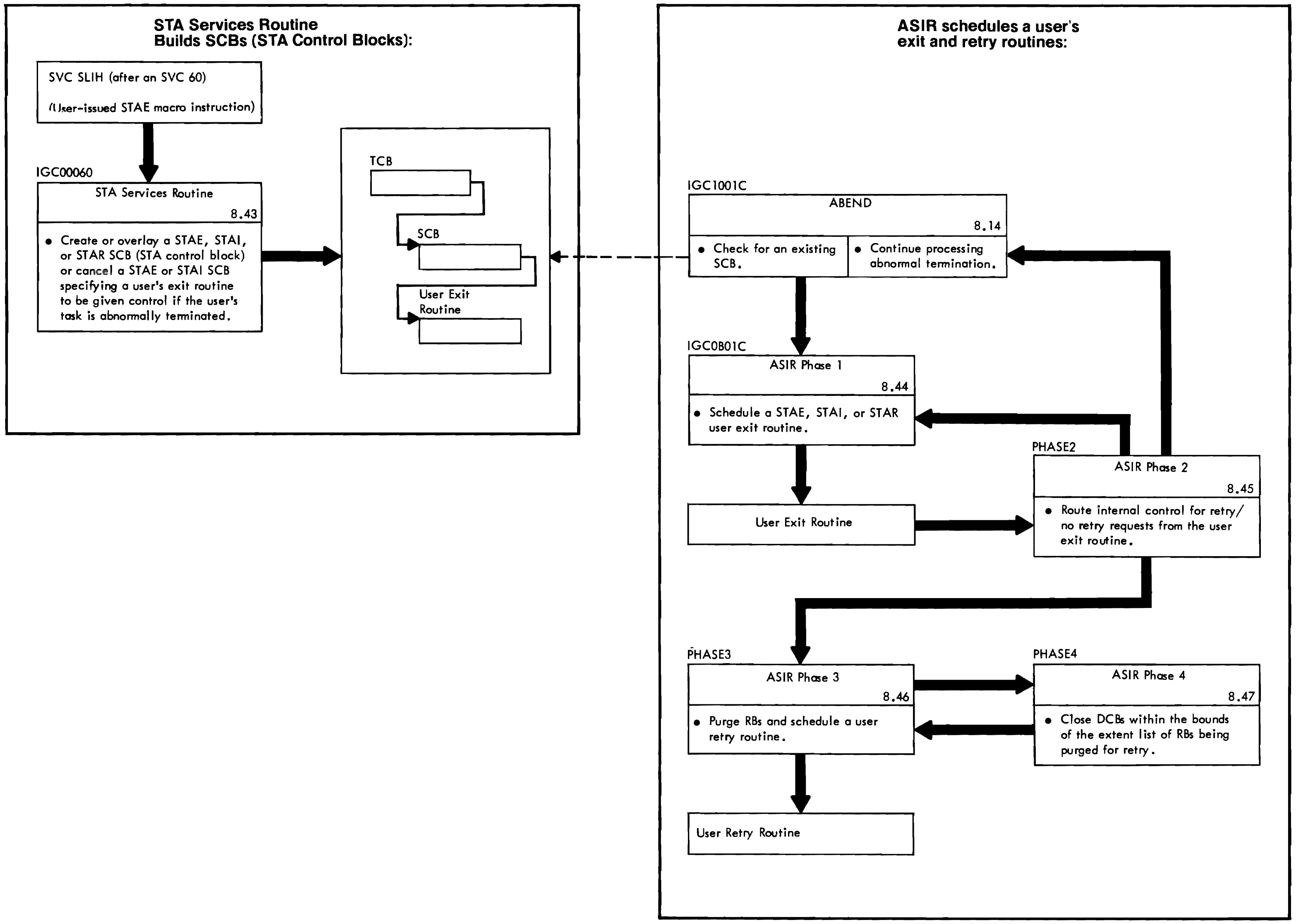

STA Services Routine
Builds SCBs (STA Control Blocks):
SVC SLIH (after an SVC 60)
(User-issued STAE macro instruction)
IGC00060
STA Services Routine
8.43
• Create or overlay a STAE, STAI, or STAR SCB (STA control block) or cancel a STAE or STAI SCB specifying a user's exit routine to be given control if the user's task is abnormally terminated.
TCB
SCB
User Exit Routine
ASIR schedules a user's
exit and retry routines:
IGC1001C
ABEND
8.14
• Check for an existing SCB.
• Continue processing abnormal termination.
IGC0B01C
ASIR Phase 1
8.44
• Schedule a STAE, STAI, or STAR user exit routine.
User Exit Routine
PHASE2
ASIR Phase 2
8.45
• Route internal control for retry/no retry requests from the user exit routine.
PHASE3
ASIR Phase 3
8.46
• Purge RBs and schedule a user retry routine.
PHASE4
ASIR Phase 4
8.47
• Close DCBs within the bounds of the extent list of RBs being purged for retry.
User Retry Routine

Diagram 8.43 (Steps 1-3) STA Services Routine

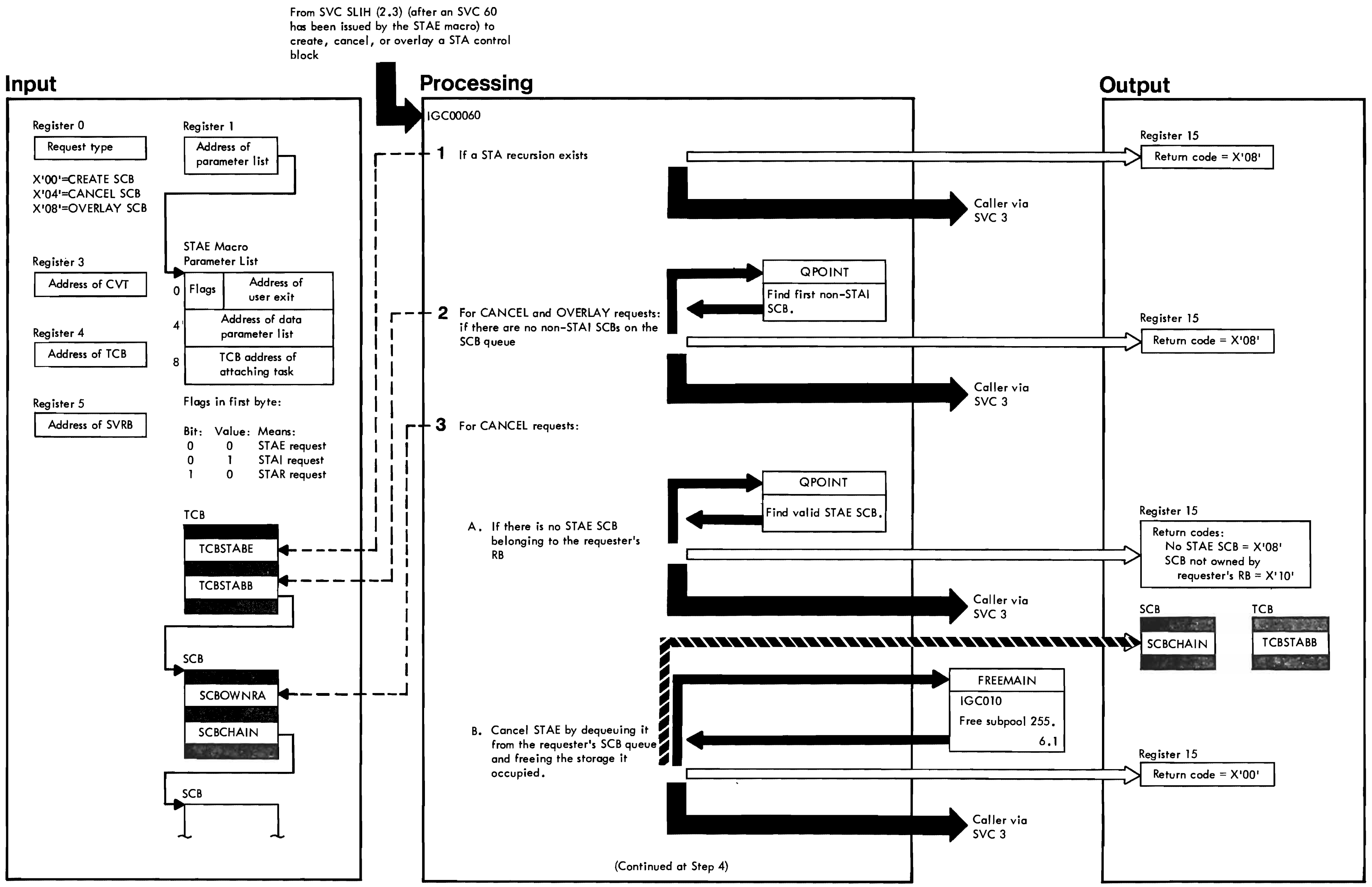
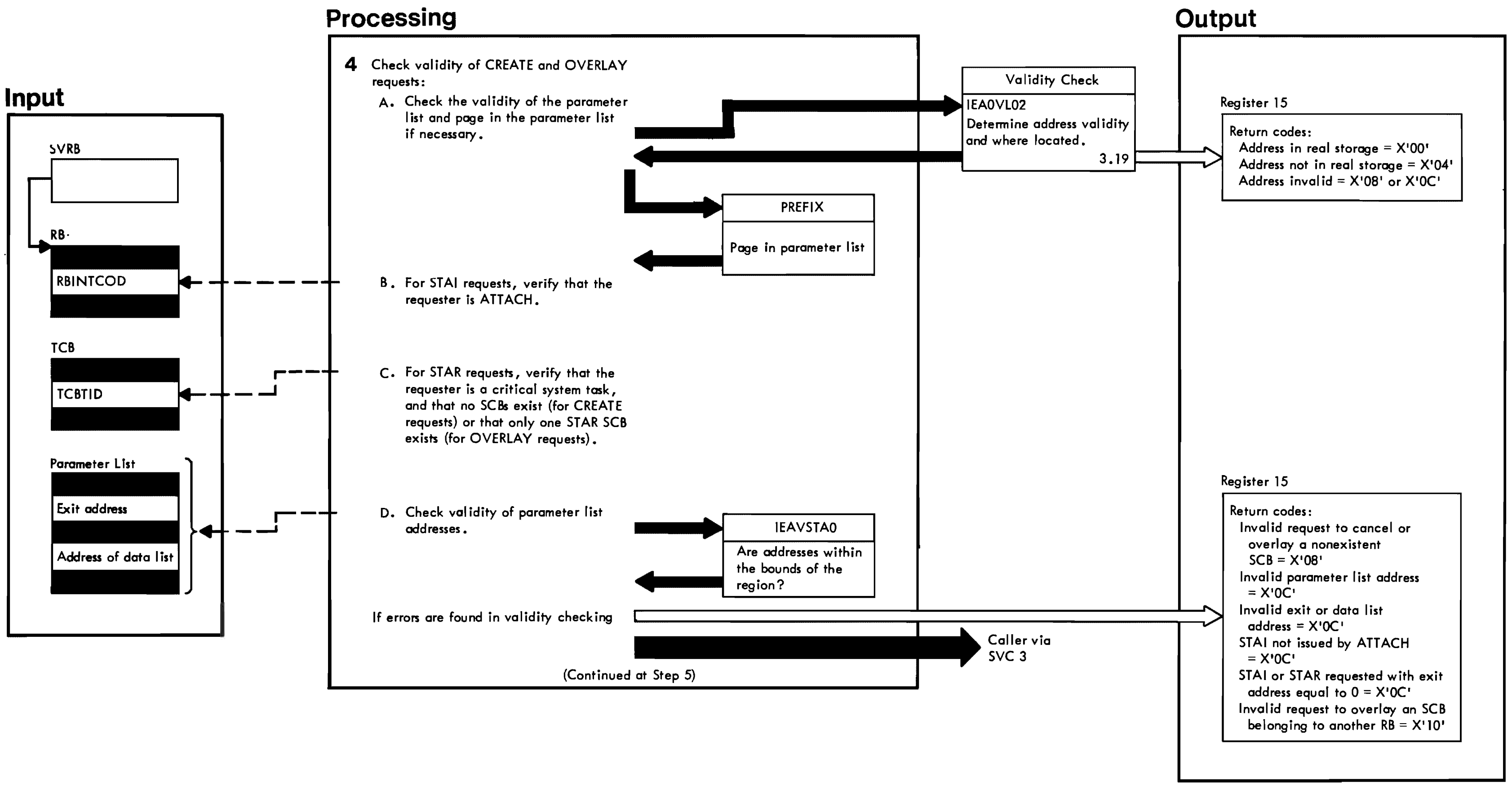


Diagram 8.43 (Steps 1-3) STA Services Routine (Module IEAVSTA0)

NOTES	ROUTINE NAME	LABEL
1 If the user's exit routine issues a STAE macro instruction, TCBSTABE is set, indicating a STA recursion. If a recursion were allowed, it would destroy the integrity of the SCB queue.	STA Services	IGC00060
2 TCBSTABB is the SCB queue pointer. If it is 0, there are no SCBs.		
3A SCBOWNRA contains the address of the RB under which the STAE macro instruction was issued. Only STAE SCBs can be canceled. There is no facility to cancel STAR or STAI SCBs.		FINDSTAE
B An SCB is dequeued by setting the preceding SCB pointer (SCBCHAIN or TCBSTABB) with the value of the canceled SCB's SCBCHAIN pointer.		QCANCEL



4 Check validity of CREATE and OVERLAY requests:

A. Check the validity of the parameter list and page in the parameter list if necessary.

B. For STAI requests, verify that the requester is ATTACH.

C. For STAR requests, verify that the requester is a critical system task, and that no SCBs exist (for CREATE requests) or that only one STAR SCB exists (for OVERLAY requests).

D. Check validity of parameter list addresses.

If errors are found in validity checking

(Continued at Step 5)

Return codes:
- Address in real storage = X'00'
- Address not in real storage = X'04'
- Address invalid = X'08' or X'0C'

Return codes:
- Invalid request to cancel or overlay a nonexistent SCB = X'08'
- Invalid parameter list address = X'0C'
- Invalid exit or data list address = X'0C'
- STAI not issued by ATTACH = X'0C'
- STAI or STAR requested with exit address equal to 0 = X'0C'
- Invalid request to overlay an SCB belonging to another RB = X'10'

Diagram 8.43 (Step 4) STA Services Routine (Module IEAVSTA0)

NOTES	ROUTINE NAME	LABEL
4 A The parameter list is paged in (if it is not in real storage) to avoid disabled missing page interruptions. The PREFIX subroutine enables the system (using MODESET) before referring to the page and forcing a missing page interruption.	STA Services	FIXPARMS
B RBINTCOD contains the ATTACH SVC number if the ATTACH routine is the requester.		
C TCBTID must be greater than 199 to represent a critical system task.		
D The STAR SCB must be the first SCB on the queue.		PCREATE

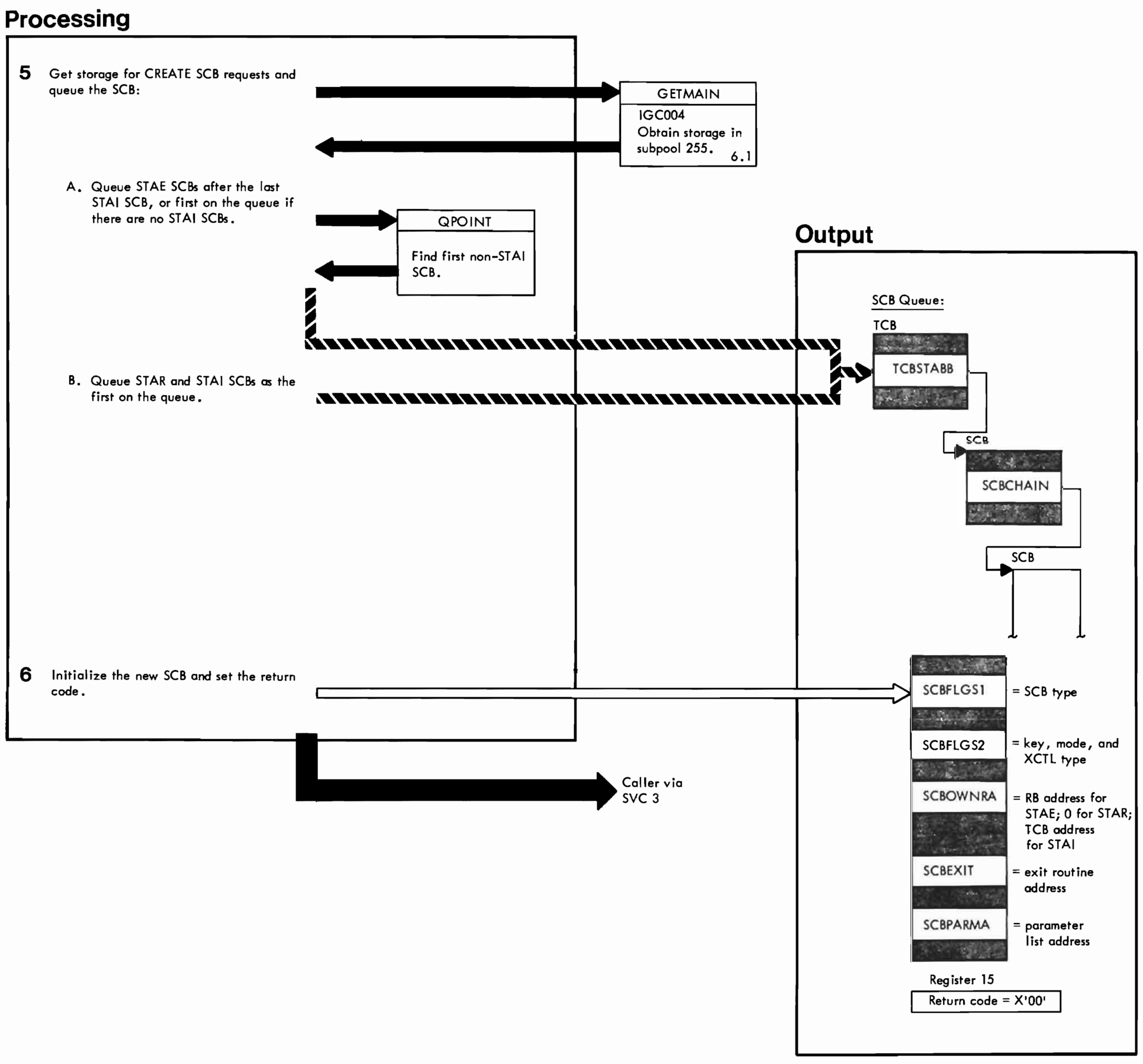

Processing
5 Get storage for CREATE SCB requests and queue the SCB:
GETMAIN
IGC004
Obtain storage in subpool 255. 6.1
A. Queue STAE SCBs after the last STAI SCB, or first on the queue if there are no STAI SCBs.
QPOINT
Find first non-STAI SCB.
B. Queue STAR and STAI SCBs as the first on the queue.
6 Initialize the new SCB and set the return code.
Caller via SVC 3
Output
SCB Queue:
TCB
TCBSTABB
SCB
SCBCHAIN
SCB
SCBFLGS1
= SCB type
SCBFLGS2
= key, mode, and XCTL type
SCBOWNRA
= RB address for STAE; 0 for STAR; TCB address for STAI
SCBEXIT
= exit routine address
SCBPARMA
= parameter list address
Register 15
Return code = X'00'

Diagram 8.43 (Steps 5-6) STA Services Routine (Module IEAVSTA0)

NOTES	ROUTINE NAME	LABEL
5 Overlay processing utilizes existing SCB storage.	STA Services	QOVERLAY
5A The QPOINT subroutine locates the last STAI SCB on an SCB queue, and the first non-STAI SCB on a queue. If there is no non-STAI SCB, QPOINT returns the address of the last STAI SCB in both output parameters. If there are no STAI SCBs, QPOINT returns the address of the first SCB in both output parameters. Otherwise, the address of the first STAE SCB or the address of the STAR SCB is returned in the second output.		FINDSTAI
B		QUESCB
6		POVERLAY

Diagram 8.44 (Steps 1-8)
ASIR Phase 1

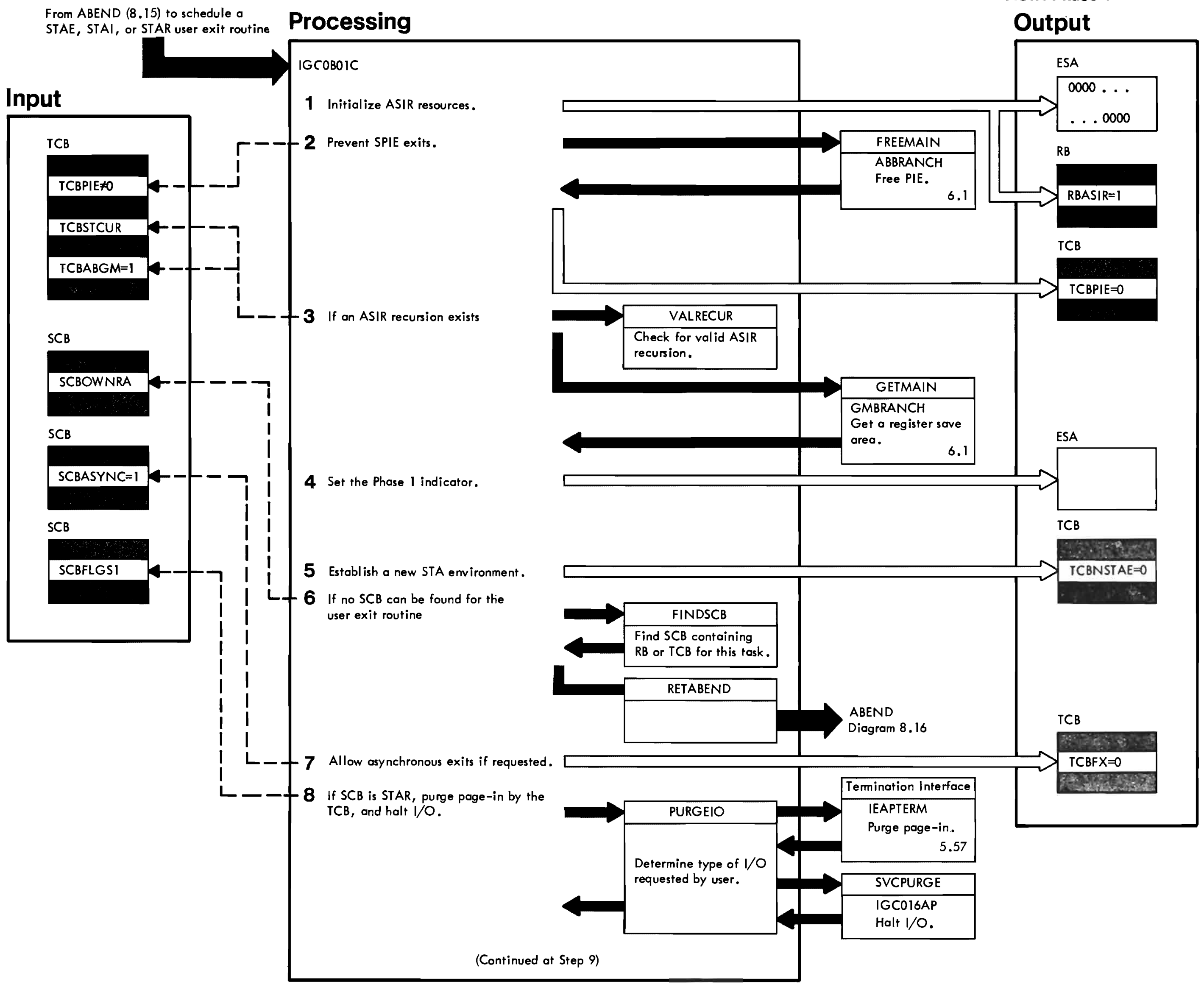

From ABEND (8.15) to schedule a STAE, STAI, or STAR user exit routine
Input
TCB
TCBPIE≠0
TCBSTCUR
TCBABGM=1
SCB
SCBOWNRA
SCB
SCBASYNC=1
SCB
SCBFLGS1
Processing
IGC0B01C
1 Initialize ASIR resources.
2 Prevent SPIE exits.
FREEMAIN
ABBRANCH
Free PIE.
6.1
3 If an ASIR recursion exists
VALRECUR
Check for valid ASIR recursion.
GETMAIN
GMBRANCH
Get a register save area.
6.1
4 Set the Phase 1 indicator.
5 Establish a new STA environment.
6 If no SCB can be found for the user exit routine
FINDSCB
Find SCB containing RB or TCB for this task.
RETABEND
ABEND
Diagram 8.16
7 Allow asynchronous exits if requested.
8 If SCB is STAR, purge page-in by the TCB, and halt I/O.
PURGEIO
Determine type of I/O requested by user.
Termination Interface
IEAPTERM
Purge page-in.
5.57
SVCPURGE
IGC016AP
Halt I/O.
(Continued at Step 9)
Output
ESA
0000 . . .
. . . 0000
RB
RBASIR=1
TCB
TCBPIE=0
ESA
TCB
TCBNSTAE=0
TCB
TCBFX=0

Diagram 8.44 (Steps 1-8) ASIR Phase 1 (Module IEAVTMOB)

NOTES	ROUTINE NAME	LABEL
1 ASIR is entered via an XCTL macro instruction from ABEND.	PHASE1	PREPHASE
2 The PIE is freed to prevent SPIE exits for program interruptions in the processing to follow: ABEND should be invoked for program interruptions.		PIETEST
3 TCBSTCUR contains the ASIR recursion flag. The following represent valid ASIR recursions. • Failure during an attempt to purge I/O with the QUIESCE option. • Failure while attempting tc initialize the STAE work area prior to exit or retry. • Failure while attempting to update the ICB restore chain when closing data sets. • Failure while attempting to update SDWA information in the event that all IOBs are dequeued from the IOB restore chain, or a failure during IOB processing. • Failure during the closing of data sets associated with an RB or RBs being purged. • Failure during the purge of enqueued resources prior to scheduling the STAR retry. • Failure during the purge of unexpired timer elements prior to scheduling the STAR retry. • TCBABGM=1 means that storage can be obtained from the WQA if the LSQA cannot satisfy a GETMAIN request.		TSTRECUR GETSAREA
4		PHASE1
5 Neither the recursion flag or the "33E ABEND" flag are reset in TCBNSTAE. This information must be saved for retry.		
6 SCBOWNRA points to the RB of the STAE SCB, or the attaching task's TCB for STAI SCBs.		
7 If SCBASYNC=1, the user has requested that asynchronous exits be allowed. ABEND sets TCBFX to 1 to prevent asynchronous exits; TCBFX is reset to 0.		GOODSCB
8 SCBFLGS1, set with SCBSTAR, indicates a STAR SCB; set with SCBNOIOP, it indicates that I/O processing should be bypassed; set with SCBHALT, it indicates that I/O should be halted. PURGEIO sets an I/O code in the ESA equivalent to the I/O codes in the parameter registers passed to the user's exit routine. (See the notes for Step 11.) For further information about the SVCPURGE routine, see <u>OS/VS I/O Supervisor Logic</u>.		CALLPURG

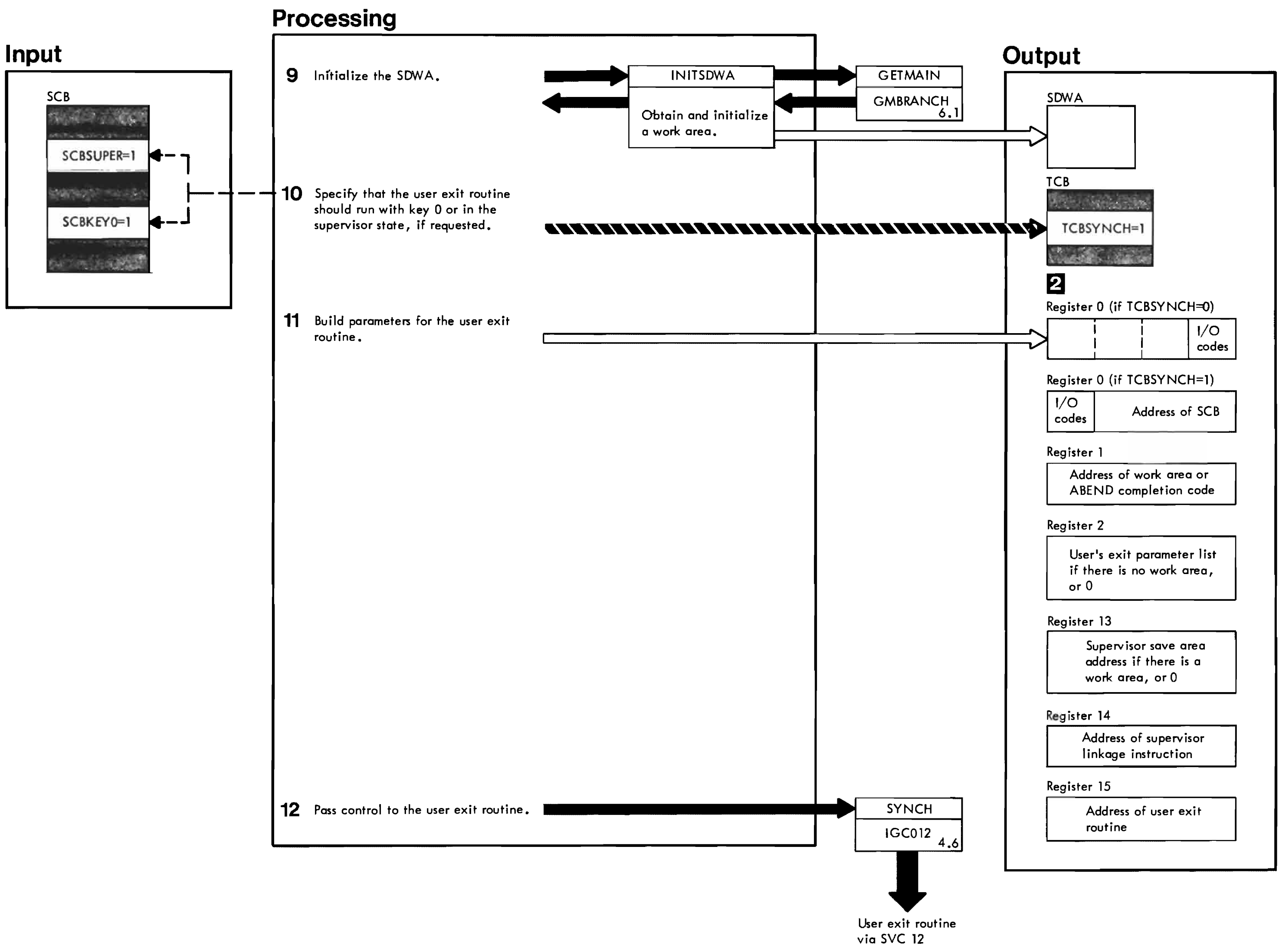
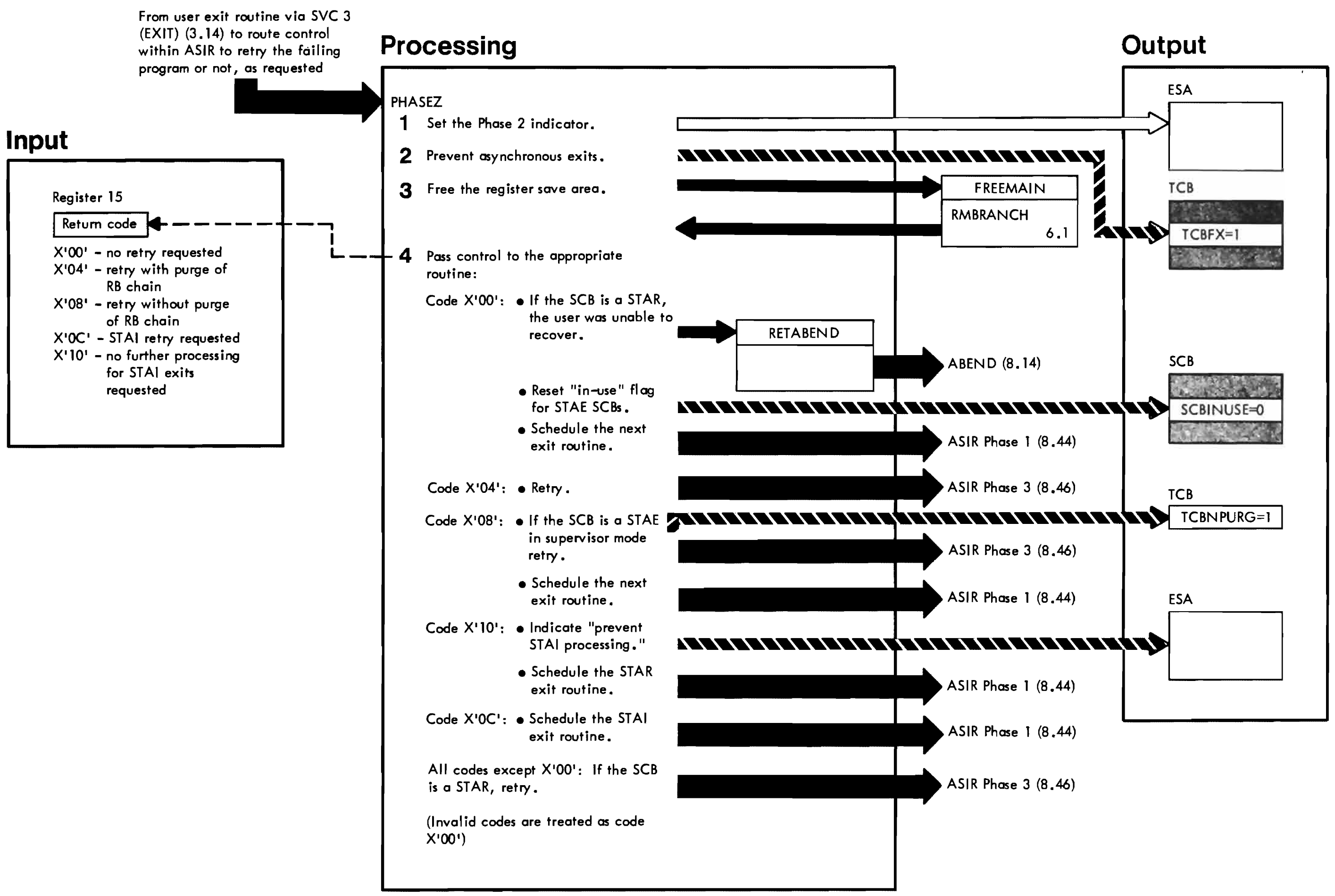

Input
SCB
SCBSUPER=1
SCBKEY0=1
Processing
9 Initialize the SDWA.
INITSDWA
Obtain and initialize a work area.
GETMAIN
GMBRANCH 6.1
10 Specify that the user exit routine should run with key 0 or in the supervisor state, if requested.
11 Build parameters for the user exit routine.
12 Pass control to the user exit routine.
SYNCH
IGC012 4.6
User exit routine via SVC 12
Output
SDWA
TCB
TCBSYNCH=1
2
Register 0 (if TCBSYNCH=0)
I/O codes
Register 0 (if TCBSYNCH=1)
I/O codes
Address of SCB
Register 1
Address of work area or ABEND completion code
Register 2
User's exit parameter list if there is no work area, or 0
Register 13
Supervisor save area address if there is a work area, or 0
Register 14
Address of supervisor linkage instruction
Register 15
Address of user exit routine

Diagram 8.44 (Steps 9-12) ASIR Phase 1 (Module IEAVTM0B)

NOTES	ROUTINE NAME	LABEL
9 • If the user is in supervisor mode, the RB address of the abnormally terminating program is entered into the SDWA. The problem program PSW field in this work area is zeroed. • If the user is in problem program mode, the name of the abnormally terminating program (or 0, if not found) is entered into the SDWA.	PHASE1	
10		SAVEINFO
11 I/O codes set in register 0 (taken from the ESA code set by PURGIO in Step 8): X'00' - Active I/O at time of ABEND was quiesced and can be restored. X'04' - Active I/O at time of ABEND was halted and can not be restored. X'08' - No active I/O at time of ABEND. X'0C' - No work area was obtained. X'10' - No I/O processing was performed.		SETPARMS
12 The SYNCH routine gives control to the user exit routine using SYNCH (SVC 102). The user exit routine must branch on register 14 (which points to an SVC 3 instruction) to return control to ASIR for a retry, or to complete ABEND processing.	SYNCH	

Diagram 8.45
ASIR Phase 2



From user exit routine via SVC 3 (EXIT) (3.14) to route control within ASIR to retry the failing program or not, as requested

Input

Register 15

Return code

X'00' - no retry requested
X'04' - retry with purge of RB chain
X'08' - retry without purge of RB chain
X'0C' - STAI retry requested
X'10' - no further processing for STAI exits requested

Processing

PHASEZ

1 Set the Phase 2 indicator.

2 Prevent asynchronous exits.

3 Free the register save area.

4 Pass control to the appropriate routine:

Code X'00': • If the SCB is a STAR, the user was unable to recover.
• Reset "in-use" flag for STAE SCBs.
• Schedule the next exit routine.

Code X'04': • Retry.

Code X'08': • If the SCB is a STAE in supervisor mode retry.
• Schedule the next exit routine.

Code X'10': • Indicate "prevent STAI processing."
• Schedule the STAR exit routine.

Code X'0C': • Schedule the STAI exit routine.

All codes except X'00': If the SCB is a STAR, retry.

(Invalid codes are treated as code X'00')

Diagram 8.45 ASIR Phase 2 (Module IEAVTM0B)

NOTES	ROUTINE NAME	LABEL
1	PHASE2	PHASE2
4 If the task abnormally terminated with a 33E condition code, a retry is not performed; control is returned to Phase 1. If control is routed to Phase 1, the work area is freed if one was obtained. The work area is retained if control is passing to Phase 3.	PHASE2	CALLROUT
Testing for:		
code X'00'	ROUTPROC	ROUTPROC
code X'04'		PRG
code X'08'		NOPRG
code X'0C'		PRG
code X'10'		QSTAIEND
TCBNPURG, when set, means that a valid no-purge request was received.		

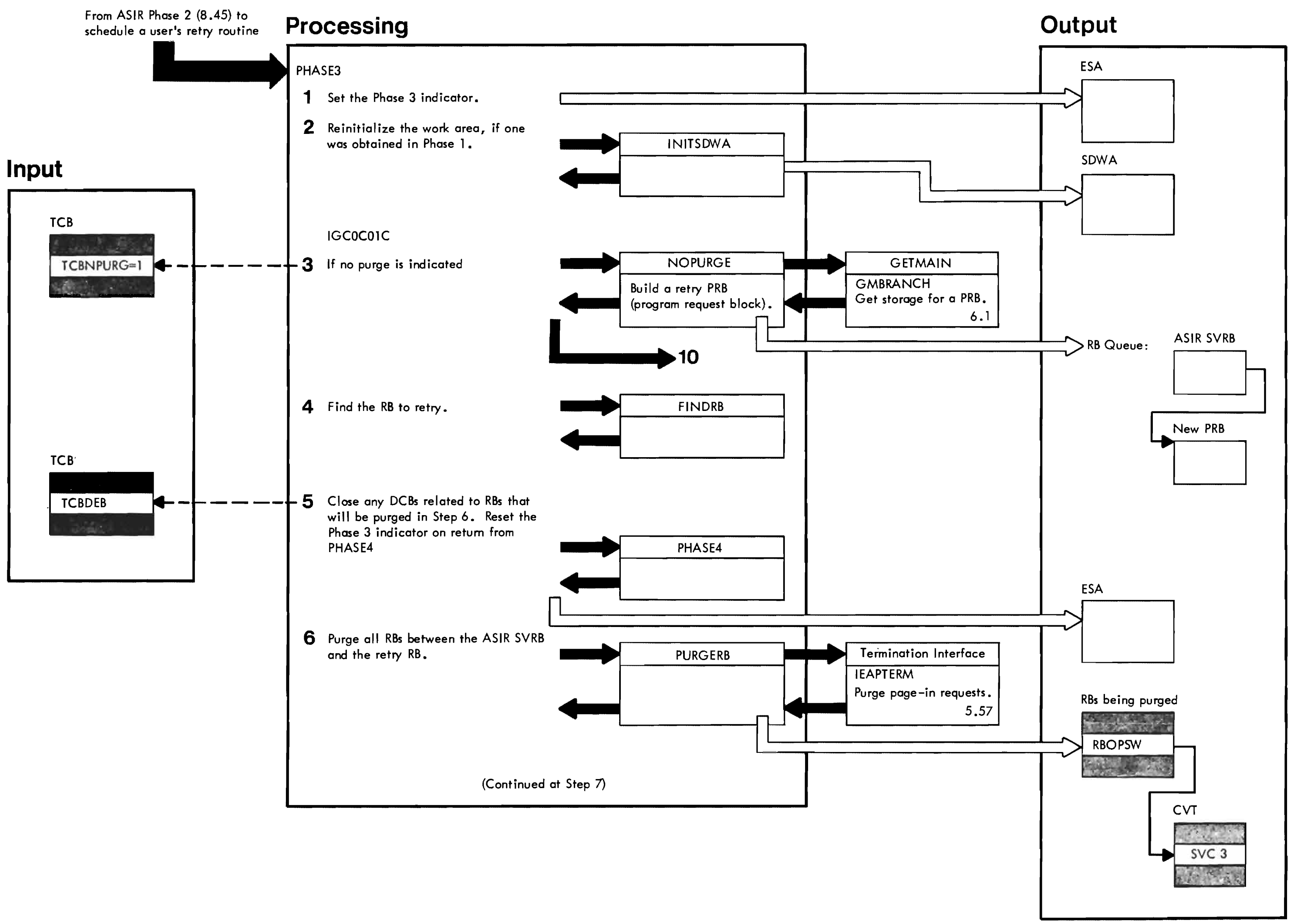

From ASIR Phase 2 (8.45) to schedule a user's retry routine
Processing
Output
Input
TCB
TCBNPURG=1
TCB
TCBDEB
PHASE3
1 Set the Phase 3 indicator.
2 Reinitialize the work area, if one was obtained in Phase 1.
INITSDWA
IGC0C01C
3 If no purge is indicated
NOPURGE
Build a retry PRB (program request block).
GETMAIN
GMBRANCH
Get storage for a PRB.
6.1
10
4 Find the RB to retry.
FINDRB
5 Close any DCBs related to RBs that will be purged in Step 6. Reset the Phase 3 indicator on return from PHASE4
PHASE4
6 Purge all RBs between the ASIR SVRB and the retry RB.
PURGERB
Termination Interface
IEAPTERM
Purge page-in requests.
5.57
(Continued at Step 7)
ESA
SDWA
RB Queue:
ASIR SVRB
New PRB
ESA
RBs being purged
RBOPSW
CVT
SVC 3

Diagram 8.46 (Steps 1-6) ASIR Phase 3 (Modules IEAVTM0B and IEAVTM0C)

NOTES	ROUTINE NAME	LABEL
1 The Phase 3 indicator is set as a debugging aid and to indicate that INITSDWA should reinitialize the work area.	PHASE3	
2 INITSDWA branches to MODESET to enable the system. It performs processing similar to that done for Phase 1 (initialization). Differences are that the user parameter list is checked for validity (if it exits; and the user is not in supervisor state). If found to be invalid (it may have been modified by the exit routine) the SDWA is freed and an indicator is set meaning that the parameter list address is invalid and no retry is allowed. Otherwise, the address of the first IOB on the restore chain (or if no I/O is restorable) is placed into word 2. Word 26 of the work area is set to point to word 2 (0 if no I/O is restorable). The name and entry point of the abnormally terminating program are placed in the work area as in Phase 1. Before returning to mainline Phase 3, INITSWA issues the MODESET SVC to disable the system.		REINIT
3 IGC0C01C (the second load module of ASIR) is entered via an XCTL macro instruction. TCBNPURG=1 means to retry without purging RBs; TCBNPURG is set in Phase 2.		TSTPRG
4 FINDRB selects the retry RB: • For STAR, RB pointing to the TCB • For STAE, RB pointed to by SCBOWNRA • For STAI, PRB whose RBLINK field points to the most recent ASIR Phase 1 SVRB. If such a PRB does not exist, the newest PRB older than the oldest non-PRB on the RB queue is used.		CALLFIND
5 TCBDEB contains the address of the first DEB, or 0, if there are no DEBs.		CLOPHASE
6 For Phase 3 processing, to ensure that correct register contents are passed to the retry routine across the SVC 3 Exit interface, PURGERB must consider the current ASIR SVRB as one of the RBs being purged.		RBPURG

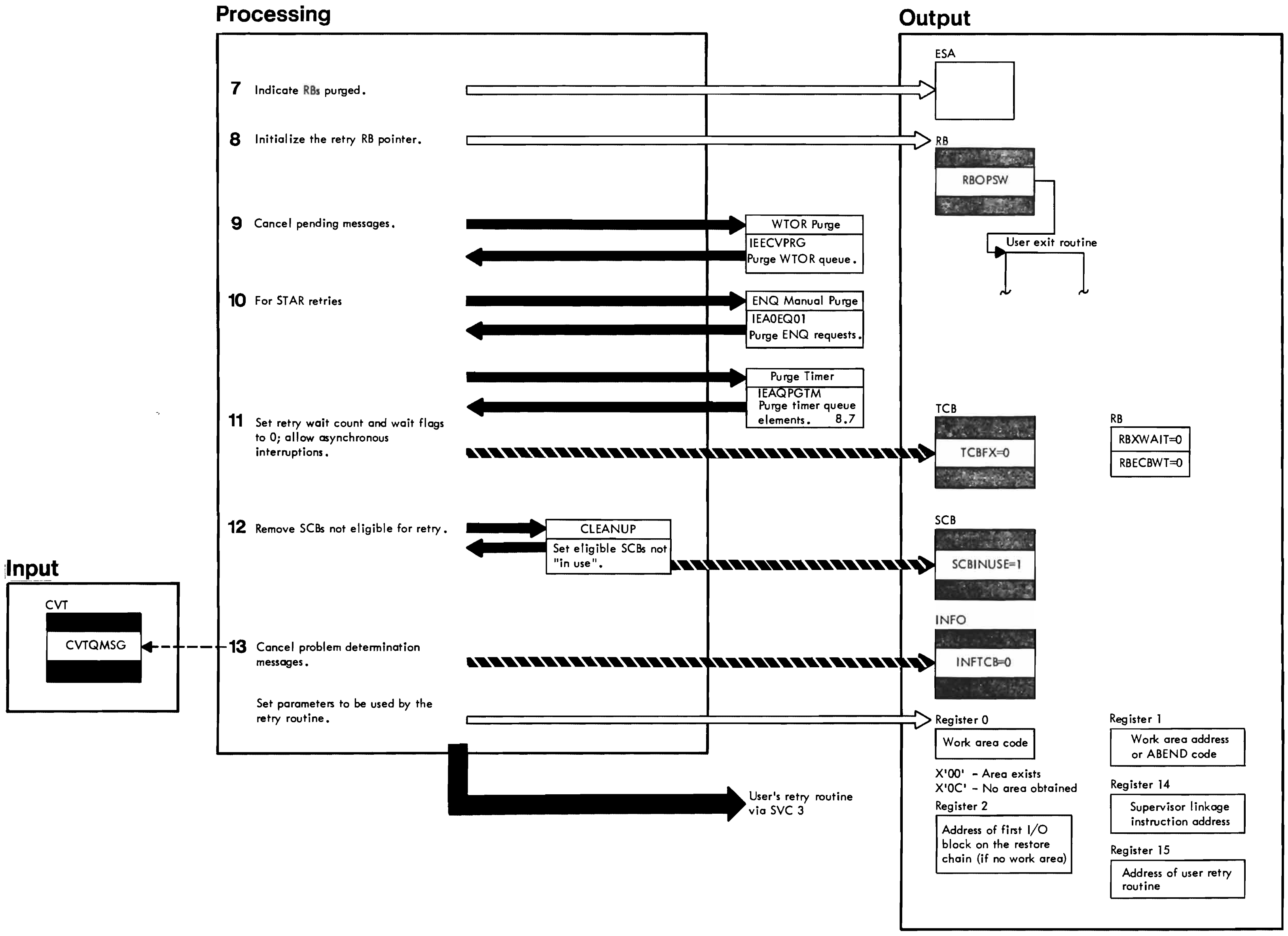
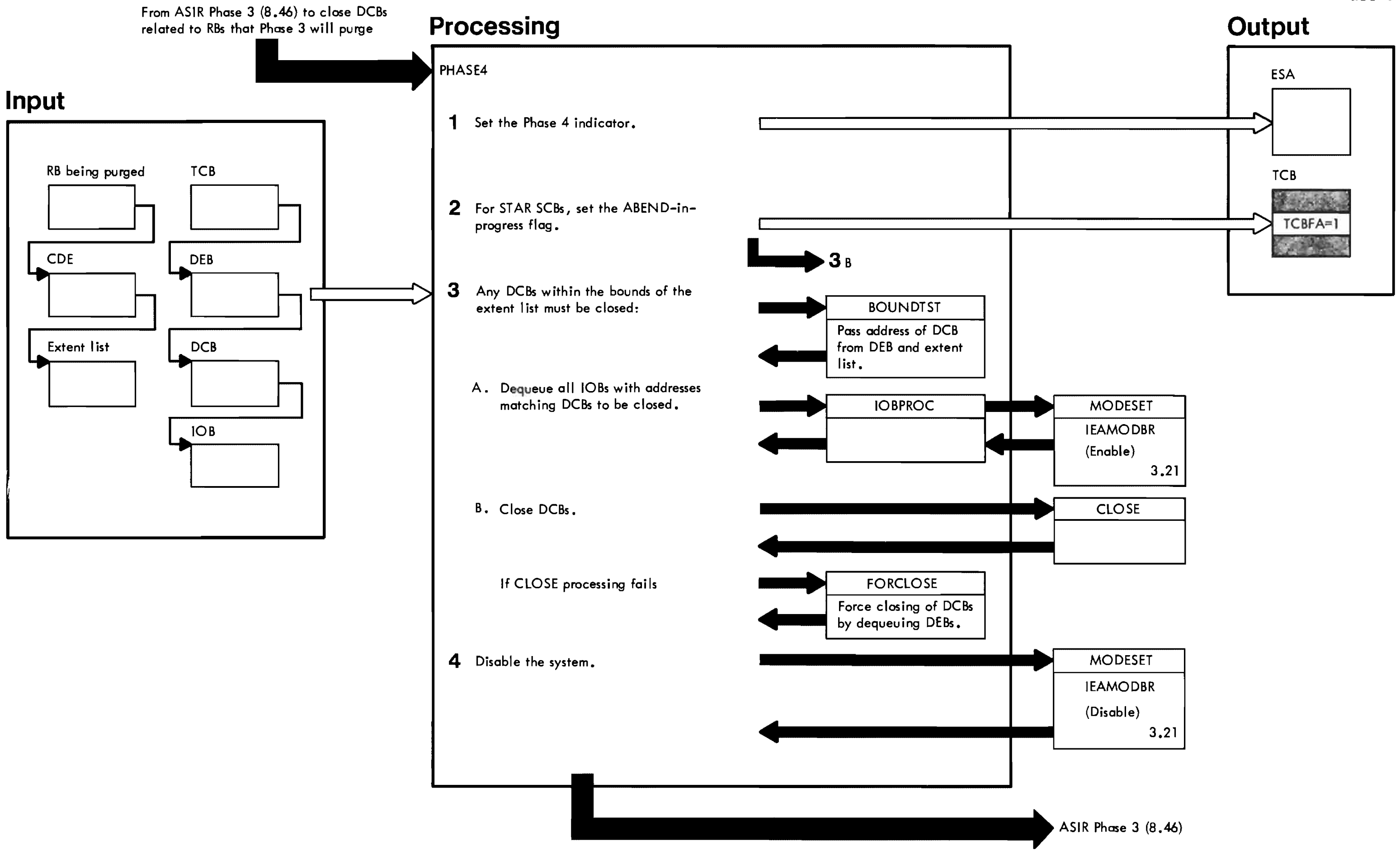

Processing
7 Indicate RBs purged.
8 Initialize the retry RB pointer.
9 Cancel pending messages.
WTOR Purge
IEECVPRG
Purge WTOR queue.
10 For STAR retries
ENQ Manual Purge
IEA0EQ01
Purge ENQ requests.
Purge Timer
IEAQPGTM
Purge timer queue elements. 8.7
11 Set retry wait count and wait flags to 0; allow asynchronous interruptions.
12 Remove SCBs not eligible for retry.
CLEANUP
Set eligible SCBs not "in use".
13 Cancel problem determination messages.
Set parameters to be used by the retry routine.
User's retry routine via SVC 3
Input
CVT
CVTQMSG
Output
ESA
RB
RBOPSW
User exit routine
TCB
TCBFX=0
RB
RBXWAIT=0
RBECBWT=0
SCB
SCBINUSE=1
INFO
INFTCB=0
Register 0
Work area code
X'00' - Area exists
X'0C' - No area obtained
Register 2
Address of first I/O block on the restore chain (if no work area)
Register 1
Work area address or ABEND code
Register 14
Supervisor linkage instruction address
Register 15
Address of user retry routine

Diagram 8.46 (Steps 7-14) ASIR Phase 3 (Module IEAVTM0C)

NOTES	ROUTINE NAME	LABEL
9 See <u>OS/VS2 Job Management Logic</u> for further information about the WTOR Purge routine.	PHASE3	WTORPRG STARPRG TIMERPRG
11 The TCBABGM flag is turned off to allow allocation only from the LSQA.		ASYNCHON
12 In CLEANUP processing, if the retry is from a STAE exit routine, the STAE SCB is dequeued from the SCB chain and the SCB storage is freed. If the retry is from a STAI exit, the in-use bit is reset in the STAI SCB. If the retry is from the oldest STAI exit, the in-use bit is reset in any STAI SCBs on the SCB queue for which the in-use bit has been previously set and not already reset. If it is a STAR retry, STAI in-use bits are reset as above in addition to resetting the STAR SCB in-use bit.		
13 Messages to be canceled were stored by type-1 SVC routines.		MSGLOOP
CVTQMSG points to a list of message entries.		
14 The work area now contains the address of the first I/O block and the address of the IOB restore chain if there is an IOB restore chain in both cases.		RETXIT



1 Set the Phase 4 indicator.

2 For STAR SCBs, set the ABEND-in-progress flag.

3 Any DCBs within the bounds of the extent list must be closed:

A. Dequeue all IOBs with addresses matching DCBs to be closed.

B. Close DCBs.

If CLOSE processing fails

4 Disable the system.

Diagram 8.47 ASIR Phase 4 (Module IEAVTMOC)

NOTES	ROUTINE NAME	LABEL
1 The Phase 4 indicator is set as a debugging aid.	PHASE4	PHASE4
2 Setting TCBFA causes the CLOSE routine to bypass its normal wait for an ECB to be posted.		ALLCLOSE
3 For STAR SCBs, _all_ DCBs are closed, not just those within the bounds of the extent list. DCBs are closed for RBs between the ASIR SVRB and the retry RB. In _IOB-PROC_ the IOBDCB field of each IOB on the restore chain is compared to the address of the DCB that is to be closed. If a match is found, that IOB is dequeued from the restore chain so that a retry routine will not later attempt to restore the IOB for a DCB which has been closed. (The CLOSE routine manipulates other IOB chains for the DCB that is being closed.) Each time a match is found on the first IOB on the restore chain, the start address of the restore chain is updated. When a match is found on the only IOB remaining on the restore chain, the "no-I/O-restorable" indicator is set in the extended save area of ASIR's SVRB and the same field in the SDWA is updated (if an SDWA exists). (Information about the IOB restore chain is passed to the user retry routine via the work area or parameter registers.)		BOUND
A IOBPROC enables the system to avoid missing page interruptions.		CALLIOBP
B For further information about the CLOSE routine, see _OS/VS OPEN/CLOSE/EOV Logic_.		CLOSEDCB DEBFRR

SECTION 9

Glossary

9

This glossary defines VS2 terms and abbreviations as they are used in this manual. This glossary does not include all terms previously established by the IBM System/360 Operating System. If you do not find the term you are looking for in this glossary, refer to the index in this book or to the IBM Data Processing Glossary, GC20-1699.

REFERENCE WORDS USED IN THE ENTRIES

The following reference words are used in this glossary:

Reference Words	Meaning
Contrast with	Refers to an opposed or substantively different term.
Preferred term is	Indicates that a term is still in use but that a preferred term exists and is defined.
Less preferably called	Indicates a synonymous term is still in use but is less prevalent than the preferred term.
See	Refers to multiple-word terms that have the same last word. The other terms may or may not be related to the entry in which the reference appears.
See also	Refers to related terms that have a similar, but not synonymous, meaning.
Synonymous with	Lists equally acceptable terms.

ACKNOWLEDGMENT TO THE AMERICAN NATIONAL STANDARDS INSTITUTE

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3.5 on Terminology and Glossary of American National Standards Committee X3.

ANSI definitions are indicated in these ways:

- An asterisk preceding the first item number indicates that all definitions for that term are ANSI definitions. In this case, the asterisk appears like this:

  term: * (1) Beginning of first definition...

- An asterisk between an item number and the beginning of a definition indicates that only the definition so marked is an ANSI definition. In this case, the asterisk appears like this:

  term: (1) * Beginning of the ANSI definition...

ABDAREA: ABDUMP work area.

ABDPL: ABDUMP subcomponent parameter list.

ABDUMP: A dump taken during abnormal termination of a task. Also the name of a routine that produces the dump.

ABDUMP subcomponent parameter list (ABDPL): An area used for communication with the TCAM, TSO, and SWA dump-formatting routines.

ABDUMP work area (ABDAREA): A work area that contains pointers, buffers, flags, and counters used by the ABDUMP routines.

ABEND: Abnormal end of task. Also the abbreviated name of a routine that performs the abnormal ending of a task and that task's incomplete subtasks.

abnormal end of task: Termination of a task prior to normal completion because of an error condition. Abbreviated ABEND.

abnormal termination: Same as abnormal end of task.

active page: A page in real storage that can be addressed.

active page queue: One of four queues of pages that are in real storage that are currently assigned to tasks. Pages on these queues are eligible for placement on the available page queue. A separate active page queue is maintained for all active pages that have a particular configuration of the reference and change bits (0,0; 0,1; 1,0; 1,1). See also available page queue, hold page queue.

address translation: The process of changing the address of an item of data or an instruction from its virtual address to its real storage address. See also dynamic address translation.

alias: (1) * An alternate label. For example, a label and one or more aliases may be used to refer to the same data element or point in a computer program. (2) An alternate entry point established within a module during linkage editor processing; an alias name is available to the control program before the module is loaded. Contrast with embedded entry point.

allocate: To assign a resource for use in performing a specific task.

allocated queue element (AQE): A VSS control block containing the size of a block of allocated storage in subpool 233, 234, 243, 244, 253, or 254.

antecedent task: Synonymous with related higher-level task.

APF: Authorized program facility.

APG: Automatic priority group.

APGCE: Automatic priority group control element.

AQE: Allocated queue element.

ASIR: ABEND/STA Interface routine.

asynchronous: Without regular time relationship; unexpected or unpredictable with respect to the execution of a program's instructions. Contrast with synchronous.

asynchronous exit queue: A queue containing queue elements that represent requests for exit routines (including data management exit routines) and requests for asynchronous control program services that must be executed under a user TCB because a disabled routine may cause a page fault. Building the asynchronous exit queue is a scheduling phase performed by the Stage 2 Exit Effector.

attaching task: Synonymous with originating task.

authorized program facility (APF): A general term for supervisor coding that limits the use of critical functions and resources to authorized users. Testing for authorization is done by TESTAUTH, a type-1 SVC routine. TESTAUTH compares an authorization code, which the linkage editor assigns to every module, against a function code whenever a task attempts to use a restricted resource or function.

automatic priority group (APG): A group of tasks at a single priority level that are dispatched according to a special algorithm that attempts to provide optimum use of CPU and I/O resources by these tasks. See also dynamic dispatching.

automatic priority group control element (APGCE): A control element used to control the dispatching of tasks in an automatic priority group.

auxiliary storage: Data storage other than real storage (and including external page storage).

auxiliary storage allocation queue: In paging supervision, a queue on which a PCB is placed to await the assignment of an external storage slot so that the page can be paged out.

auxiliary storage management: A general name used for routines in the paging supervisor that control external page storage.

available frame count: A count of page frames that are ready for reassignment.

available page queue: A queue of the pages whose real storage is currently available for allocation to any task. See also active page queue, hold page queue.

basic control (BC) mode: A mode in which the features of a System/360 computing system and additional System/370 features, such as new machine instructions, are operational on a System/370 computing system. See also extended control (EC) mode.

BC mode: Basic control mode.

BLDL table: A table of track addresses for a user-specified list of modules on the link library (SYS1.LINKLIB). The purpose of the table is to reduce the time required to find the listed modules on SYS1.LINKLIB. In VS2, the user can specify that this table is to be either paged or fixed; when the table is on fixed pages in lower real storage, it is called a fixed BLDL table.

blocking: Combining two or more logical records into one physical record.

branch-entry fix-delay queue: In paging supervision, a queue on which fix requests (received through branch entries to the FIX/LOAD routine) are placed when they cannot be processed because the amount of fixable real storage has fallen below a specified minimum.

BSAM: Basic sequential access method.

CCW translation: Preferred term is channel program translation.

CDE: Contents directory entry.

change bit: A bit associated with a page in real storage; the change bit is turned on by hardware whenever the associated page in real storage is modified. In VS2, there is a change bit in each of two storage keys associated with each frame; if either bit is on, the entire page is considered to have been modified.

channel program queue: A queue is the nucleus containing channel programs used by the paging supervisor for paging operations.

channel program queue element (CPQE): A control element in the nucleus that contains a channel program used by the paging supervisor.

channel program translation: In a channel program, replacement by the software of virtual addresses with real addresses. Less preferably called CCW translation.

clock comparator feature: A System/370 CPU feature used to cause an external interruption when the time-of-day clock reaches or passes a specified value and the CPU is enabled for clock comparator interruptions. A value is placed in the clock comparator, and when the time-of-day clock reaches that value, the interruption occurs.

clock comparator queue: The queue of timer queue elements (TQEs) representing requests for REAL and WAIT timing. Each TQE contains an interval; the interval contained in the TQE at the head of this queue is placed in the clock comparator.

communications vector table (CVT): A major table in the system nucleus that contains addresses and other data needed and used by various supervisor routines.

contents directory: A group of queues that indicate the routines that are present in virtual storage. Includes the link pack area queue, job pack area queue, and time sharing link pack area queue.

contents directory entry (CDE): A control entry that represents a module in the job pack area. The entry describes the module and contains a pointer to its entry point. If an embedded entry point was established in the module by an IDENTIFY macro instruction, there are two CDEs for the module. One, the major CDE, contains the main entry-point address; the other, the minor CDE, contains the entry-point address for the embedded entry point.

control registers: A set of registers used for operating system control of relocation, priority interruption, program event recording, error recovery, and masking operations. Contrast with general-purpose registers.

CPQE: Channel program queue element.

CPU timer: A System/370 timing device used for TASK timing. It is a 64-bit decrementing binary counter. On the basis of a program request, a binary time interval is placed in the counter. A one is subtracted from bit position 51 of the counter every microsecond. An external interruption occurs when the value placed in the CPU timer reaches or becomes less than zero if the CPU is enabled for CPU timer interruptions.

CPU timer queue: The queue of timer queue elements (TQEs) representing requests for TASK timing. Each TQE contains an interval, and the interval contained in the TQE at the head of this queue is placed in the CPU timer.

CVT: Communications vector table.

default device: A paging device from which a page slot is to be assigned for a page-out operation when a particular device has not been specified in the page-out request.

deferred allocation: A condition in which the allocation of pages for a region must be delayed because all needed pages are not immediately available.

delayed posting queue: In paging supervision, a queue on which a PCB is placed when a paging operation has been completed but the completion cannot be posted immediately because the system is handling a disabled page fault.

demand paging: Transfer of a page from external page storage to real storage at the time it is needed for execution.

dequeue: (1) To remove a program from a list of programs awaiting use of a system service or resource. (2) To remove a control block or data entry from a queue of control blocks or data entries. (3) Contrast with enqueue.

descendant task: Synonymous with related subtask.

descriptor queue element (DQE): A VSS control block that describes a storage block of 4K or more bytes within a subpool.

device number: A part of an external page address that refers to a particular paging device; together with a group number and a slot number, it identifies the location of a page in external page storage.

directed page-out: A page-out operation in which the page slot is to be assigned from a paging device specified in the page-out request (rather than from a default device).

disabled: Pertaining to a state of the CPU that prevents the occurrence of certain types of interruptions.

disabled code: A section of code that cannot be interrupted by specific types of interruptions.

disabled page fault: A page fault that occurs when I/O and external interruptions are disallowed by the CPU.

dispatch: To place into execution. See also task switching.

dispatching priority: A number assigned to each task, used to determine the order in which that task will use the CPU in relation to other tasks in the system. See also limit priority.

dormant state: A state in which the active pages of a job have been paged out.

DQE: Descriptor queue element.

DSS: Dynamic support system.

dump: * (1) To copy the contents of all or part of a storage, usually from an internal storage into an external storage. (2) A process as in (1). (3) The data resulting from the process as in (1).

dynamic address translation (DAT): (1) The process of changing a virtual storage address to a real storage address during execution of an instruction. See also address translation. (2) A hardware feature that performs the translation.

dynamic allocation: Assignment of system resources to a program at the time the program is executed rather than at the time it is loaded into storage.

dynamic area: The portion of virtual storage that is divided into regions that are assigned to job steps and system tasks. See also pageable dynamic area, non-pageable dynamic area. Contrast with non-dynamic area.

dynamic dispatching: A facility that assigns priorities to tasks within an automatic priority group to provide optimum use of CPU and I/O resources.

dynamic dump: A dump that is performed during the execution of a computer program.

dynamic support system (DSS): An interactive debugging facility that allows authorized maintenance personnel to monitor and analyze events and alter data. Information for DSS is included for planning purposes only.

ECB: Event control block.

EC mode: Extended control mode.

embedded entry point: An alternate entry point established within a module by means of the IDENTIFY macro instruction after a module has been loaded into virtual storage. Contrast with alias.

enabled: Pertaining to a state of the CPU that allows the occurrence of certain types of interruptions. Synonymous with interruptable.

enabled code: Code that can be interrupted by the CPU.

enabled page fault: A page fault that occurs when I/O and external interruptions are allowed by the CPU.

enqueue: (1) To add a program to a list of programs awaiting use of a system service or resource. (2) To add a control block or data entry to a queue of control blocks or data entries. (3) Contrast with dequeue.

event control block: A control block used to represent the status of an event.

extended control (EC) mode: A mode in which all the features of System/370 computing system, including dynamic address translation, are operational. See also basic control (BC) mode.

extent list (XTLST): A control element that shows the address and length of each module in virtual storage.

external page: A page located on external page storage.

external page address: An address that identifies the location of a page in a page data set. In VS2, the address consists of a relative device number, a relative group number, and a relative slot number.

external page storage: The portion of auxiliary storage that is used to contain pages.

external page table (XPT): An extension of a page table that identifies the location on external page storage of each page in that page table.

external page table entry (XPTE): An entry, in an external page table, that associates a virtual storage page with the actual page on an auxiliary storage device.

FBQE: Free block queue element.

fix ownership element (FOE): A control element that represents a page that has been fixed by the task to whose TCB the FOE is chained.

fixed: In OS/VS, not capable of being paged out. See also long-fixed, normal-fixed, short-fixed, and page fixing.

fixed BLDL table: A BLDL table that the user has specified to be fixed in the lower portion of real storage.

fixing: See page fixing.

fixed link pack area: An extension of the link pack area that occupies fixed pages in the lower portion of real storage. A routine in the fixed LPA is used in preference to a copy of the same routine in the pageable LPA. The routines in the fixed LPA are densely packed without regard for page boundaries.

fixed page: A page in real storage that is not to be paged out.

FLIH: First-level interruption handler.

FOE: Fix ownership element.

FQE: Free queue element.

frame: Same as page frame.

frame number: In VS2, the part of a real storage address needed to refer to a frame. See also page number.

frame table: Same as page frame table.

frame table entry (FTE): An entry in the page frame table that describes how a frame is being used.

free block queue element (FBQE): A VSS control block that describes 4K blocks of storage not yet assigned to a subpool.

free queue: In paging supervision, a queue on which a PCB is placed when a request for a paging service has been completely satisfied and this PCB is no longer needed. The PCBs on this queue are available for use in handling other requests for paging services.

free queue element (FQE): A VSS control block that describes the unallocated storage within the space defined by a descriptor queue element (DQE).

FTE: Frame table entry.

general-purpose registers: A set of registers available to any program running in a CPU. Contrast with control registers.

generalized trace facility (GTF): A tracing program that monitors and records system events. It may be turned on and off by the operator and manipulated by the termination routines.

GOVRFLB: The primary VSS control block. It defines SQA and LSQA quickcell requirements and contains data needed for initial supervision of virtual storage.

group: See slot group.

group number: In VS2, a part of an external page address that refers to a slot group; together with a device number and a slot number, it identifies the location of a page in external page storage.

GTF: Generalized trace facility.

hold page queue: A queue to which pages in real storage are initially assigned through operations such as page-in or page reclamation. See also active page queue, available page queue.

INFOLIST: Type-1 SVC message table.

initial program loader (IPL): A program that loads the control sections of the control program into real storage.

initial program loading: A combined hardware/software operation that clears real storage and loads the control sections of the control program into real storage. The hardware portion of the operation starts a read operation from a designated input device; the IPL program then uses a bootstrap program to load itself into real storage. Abbreviated IPL.

initiating task: The job management task that controls the selection of a job and the preparation of the steps of the job for execution. Less preferably called initiator task.

interception: In paging supervision, the process of seizing a page frame that was being used for a non-V=R purpose and is needed to complete a requested V=R region. The PFTE (page frame table entry) for the frame was previously marked so that the frame would be seized when it was released from its non-V=R use. See also V=R intercepted page.

interregion posting: The process by which the POST routine indicates to a swapped-out TSO routine that an awaited event is complete. The POST routine passes control to TSO's TSIP (Time Sharing Interface Program) to swap in the user. Execution of the POST routine is then requested by the Region Control Task of TSO.

interruption queue element (IQE): A control element used by the Stage 2 and Stage 3 Exit Effectors to schedule the execution of an end-of-task exit routine (ETXR).

interruption request block (IRB): A control block that represents a routine that must be executed because of the occurrence of an asynchronous interruption or event.

interval: A predetermined time period.

interval read count: A count of the number of page-in operations that are requested during a specified interval. The count is calculated by the task disablement algorithm in paging supervision.

interval reclaim count: A count of the number of pages that, during a specified interval, have been reclaimed from the available page queue or have been reclaimed during preliminary page-out processing. The count is calculated by the task disablement algorithm in paging supervision.

interval swap-in count: A count of the number of page-in requests that were required by swap-in operations during a specified interval. The count is calculated by the task disablement algorithm in paging supervision.

invalid page: A page that cannot be directly addressed by the dynamic address translation feature of the CPU.

I/O active queue: In paging supervision, a queue on which a PCB is placed whenever a page-out or page-in operation has been started to satisfy a request represented by the PCB.

IPL: Initial program loader.

IRB: Interruption request block.

IQE: Interruption queue element.

JCT: Job control table.

JFCB: Job file control block.

job: A collection of related problem programs, identified in the input stream by a JOB statement followed by one or more execute (EXEC) and data definition (DD) statements.

job pack area (JPA): The two subpools (251 and 252) in a given region of virtual storage into which executable programs are loaded.

job pack area queue (JPAQ): A queue containing the contents directory entries (CDEs) for problem programs running in the job pack area subpools of a region.

job step: (1) * The execution of a computer program explicitly identified by a job control statement. A job may specify that several job steps be executed. (2) A unit of work associated with one processing program or one cataloged procedure and related data. A job consists of one or more job steps.

job-step task: (1) A task that is initiated by an initiator/terminator in accordance with specifications in an execute (EXEC) statement. (2) A subtask of a job-step task whose job-step pointer points to itself.

limit priority: A priority specification associated with each task, representing the highest dispatching priority that the task can assign to itself or to any of its subtasks. See also dispatching priority.

link pack area (LPA): An area of virtual storage that contains selected reenterable and serially reusable routines that are loaded at NIP time and can be used concurrently or serially by all tasks in the system. In VS2, a link pack area can consist of a pageable LPA only, or of a pageable LPA and a fixed LPA. See also pageable link pack area and fixed link pack area.

link pack area directory: A directory that contains an entry (LPDE) for each entry point in all modules in the link pack area.

link pack area library (SYS1.LINKLIB): A partitioned data set that contains the modules specified to be in the link pack area.

link pack area queue: A queue that contains a contents directory entry (CDE) for each link pack area module currently in use, for each module in the link pack update area, and for each module in the fixed link pack area. In searching for a requested module, the system searches this queue before it searches the link pack area directory.

link pack update area: An area in virtual storage that contains modules that are additions to or replacements for link pack area modules for the current IPL.

LLE: Load list element.

load list: The chain of load list elements that represent the modules that have been loaded into a task's job pack area.

load list element (LLE): A control element created for each module loaded into a region's job pack area by the LOAD routine. The LLEs for each task in a job step are chained together to form the load list for that task. Each LLE points to a contents directory entry for the loaded module.

local system queue area (LSQA): One or more segments associated with each virtual storage region that contains job-related system control blocks.

lock/unlock facility: A supervisor facility that controls the execution of instruction strings when a disabled page fault occurs.

long-fixed: Pertaining to a request to fix one or more pages for a relatively long time.

LPA: Link pack area.

IPDE: Link pack directory entry.

LSQA: local system queue area.

low threshold: In paging supervision, the minimum number of page frames that can be available before the system takes action to acquire additional unoccupied frames.

message buffer: A structured buffer area used for building and writing system messages.

migration: See page migration.

migration PCB: A PCB representing a page to be migrated.

migration queue: In paging supervision, a queue on which a PCB is placed whenever a page must be moved from a primary to a secondary paging device to make paging more efficient.

missing page interruption: See page fault.

must-complete status: See step must-complete status and system must-complete status.

NIP: Nucleus initialization program.

nondirected page-out: A page-out operation in which the page slot is to be assigned from a default device (rather than from a particular device specified in the page-out request).

nondynamic area: The area of virtual storage assigned to the resident portion of the control program (the nucleus and the link pack area). Contrast with dynamic area.

nonpageable dynamic area: An area of virtual storage whose virtual addresses are identical to real addresses; it is used for programs or parts of programs that are not to be paged during execution. Less preferably called V=R dynamic area.

nonpageable region: A subdivision of the nonpageable dynamic area that is allocated to a job step or system task that is not to be paged during execution. In a nonpageable region, each virtual address is identical to its real address. Less preferably called V=R region.

normal-fixed: Pertaining to a request to fix one or more pages for an average period of time.

nucleus initialization program (NIP): The program that initializes the resident control program. It allows the system operator to request last-minute changes to certain options specified during system generation. Abbreviated NIP.

originating task: The task that is attaching or did attach a particular subtask. Synonymous with attaching task.

overlay segment: A user-determined portion of a program that has been divided so that the portions can be consecutively loaded into virtual storage and executed. Contrast with segment.

overlay segment table (SEGTAB): A table used by the overlay supervisor to record the use of overlay segments. Contrast with segment table (SGT) used in dynamic address translation.

page: (1) A fixed-length block of instructions, data, or both, that can be transferred between real storage and external page storage. In VS2, a page consists of 4K bytes. (2) To transfer instructions, data, or both between real storage and external page storage.

page control block (PCB): A control block that indicates the status of a paging request, and is used to control the movement of a paging request among the various modules of the paging supervisor.

page data set: A data set in external page storage, in which pages are stored.

page fault: A program interruption that occurs when a page that is marked "not in real storage" is referred to by an active page. Synonymous with page translation exception.

page fixing: Marking a page as nonpageable so that it remains in real storage.

page frame: A block of real storage that can contain a page. Synonymous with frame.

page frame table (PFT): A table that contains an entry for each frame. Each frame table entry describes how the frame is being used.

page frame table entry (PFTE): A control element that reflects the status of a page frame.

page-in: The process of transferring a page from external page storage to real storage.

page I/O initiation queue: In paging supervision, a queue on which a PCB is placed to await the starting of an I/O operation required for the page-out or page-in request represented by the PCB.

page migration: The transfer of pages from a primary paging device to a secondary paging device to make more space available on the primary paging device.

page number: The part of a virtual storage address needed to refer to a page. See also frame number.

page-out: The process of transferring a page from real storage to external page storage.

page reclamation: The process of making addressable the contents of a page in real storage that has been marked invalid. Page reclamation can occur after a page fault or after a request to fix or load a page.

page table (PGT): A table that indicates whether a page is in real storage and correlates virtual addresses with real storage addresses.

page table entry (PTE): An element in the page table that represents a page in real storage.

page translation exception: A program interruption that occurs when a virtual address cannot be translated by the hardware because the invalid bit in the page table entry for that address is set. Synonymous with page fault, program interruption code 17. See also segment translation exception, translation specification exception.

page vector table (PVT): The central control block for the paging supervisor.

page wait: A condition in which the active request block for a task is placed in a wait state while a requested page is located in real storage or is brought into real storage.

pageable dynamic area: An area of virtual storage whose addresses are not necessarily identical to real addresses; it is used for programs that can be paged during execution. Less preferably called V=V dynamic area.

pageable region: A subdivision of the pageable dynamic area that is allocated to a job step or system task that can be paged during execution. Less preferably called V=V region.

paging: The process of transferring pages between real storage and external page storage.

paging device: A direct access storage device on which pages (and possibly other data) are stored.

paging device information table entry (PDITE): A control entry used by the paging supervisor to identify the characteristics and status of a paging device. There is one PDITE for each paging device.

paging device table entry (PDTE): A control entry used by the paging supervisor to locate specific pages on a paging device.

paging rate: The average number of page-ins and page-outs per unit of time.

paging supervisor: A part of the supervisor that allocates and releases real storage space (page frames) for pages, and initiates page-in and page-out operations.

partition queue element (PQE): A VSS control block that heads a chain of control blocks that describe unallocated storage space for an entire region, an LSQA, or the SQA.

PCB: Page control block.

PCI: Program-check interruption.

PDITE: Paging device information table entry.

PDTE: Paging device table entry.

PER: Program event recording.

PFT: Page frame table.

PFTE: Page frame table entry.

PICA: Program interruption control area.

PIE: Program interruption element.

PGT: Page table.

PQE: Partition queue element.

PRB: Program request block.

primary paging device: An auxiliary storage device that is used in preference to secondary paging devices for paging operations. Portions of a primary paging device can be used for purposes other than paging operations.

priority: A dispatching priority and a limit priority for each task resides in the task's TCB. The dispatching priority determines the appropriate position of the TCB in the TCB queue; the limit priority is used by the CHAP routine to determine the maximum increase possible for the dispatching priority. See also dispatching priority and limit priority.

problem state: A state during which the CPU cannot execute input/output and other privileged instructions. Contrast with supervisor state.

program event recording (PER): A hardware feature used to assist in debugging programs by detecting program events.

program interruption code 16: Same as segment translation exception.

program interruption code 17: Same as page translation exception.

program interruption code 18: Same as translation specification exception.

program interruption control area (PICA): A control block used to control the executing of a user exit routine that was specified in a SPIE macro instruction.

program interruption element (PIE): A control element created when a SPIE macro instruction is used to specify that an exit routine is to process a particular type of program-check interruption.

program request block (PRB): A request block that represents a nonsupervisory module or routine that is to be executed on behalf of a task.

PTE: Page table entry.

purge: To remove program elements or control information from a system.

PVT: Page vector table.

QCB: Queue control block.

QCDBLK: Quickcell descriptor block.

QEL: Queue element.

queue control block (QCB): A control block that represents a set of resources (major QCB) or a single resource (minor QCB).

queue element (QEL): A queue element used to represent one request for a particular system resource. Queue elements are used in handling the serial use of a system resource by different programs.

quickcell: A small unit of storage (ranging from 8 to 240 bytes) in the system queue area or a local system queue area that is allocated upon request for a short-lived control block. Allocation of a quickcell is faster than normal GETMAIN allocation, and use of quickcells reduces storage fragmentation.

quickcell area: A portion of the system queue area or a local system queue area that contains quickcells.

quickcell descriptor block (QCDBLK): A VSS control block that describes a quickcell area in the SQA or an LSQA.

quiescing: (1) The process of bringing a device, an activity, or a system to a halt by rejection of new requests for work. (2) The process of bringing a multiprogramming system to a halt by rejection of new jobs.

RB: Request block.

real address: The address of a location in real storage.

real storage: The storage of a System/370 computing system from which the central processing unit can directly obtain instructions and data, and to which it can directly return results.

real storage allocation queue: In paging supervision, a queue on which a PCB is placed whenever a page frame is needed to satisfy a page-in request but no frame is currently available.

REAL timing option: An option in the STIMER macro instruction. The option specifies that the requested time interval is to be decremented continuously, whether or not the task is executing. Contrast with TASK timing option. See also WAIT timing option.

recursion: Reentry into a routine as a result of a failure in the routine's processing. The routine is entered at a point subsequent to the point of failure.

recursive: Pertaining to a process in which each step makes use of the results of earlier steps.

reference bit: A bit associated with a page in real storage; the reference bit is turned on by hardware whenever the associated page in real storage is referred to (read or stored into). In VS2, there is a reference bit in each of two storage keys associated with each page frame.

region: See virtual storage region.

related higher-level task: On the subtask queue for a job step, any task that was attached at a higher level and is connected through the task chain to a particular task being discussed. Synonymous with antecedent task.

related PCB: A PCB whose request for a particular page can be associated with an already existing PCB that is accessing that same page.

related subtask: On the subtask queue for a job step, any task that was attached at a lower level and is connected through the task chain to a particular task being discussed. Synonymous with descendant task.

relocation feature: Preferred term is dynamic address translation.

relocation hardware: Preferred term is dynamic address translation.

reply queue element (RQE): A control element used to schedule asynchronous exit routines for data management.

request block (RB): A control block that represents a module or routine to be executed on behalf of a task. See also interruption request block, program request block, system interruption request block, supervisor request block, and task interruption request block.

reserve replenish queue: In paging supervision, a queue that is readied whenever a page must be freed to replace a page that was reserved for emergency SQA or LSQA use and has now been allocated to one of those areas. The queue may also be activated because the available page count was zero and the page replacement algorithm could not be invoked immediately.

root page control block (root PCB): A control block then is used to control the posting for a set of required operations on a defined group of pages when the posting cannot take place until all of the operations have been completed for all of the pages.

root PCB: Root page control block.

RQE: Reply queue element.

same-level tasks: Two or more subtasks that were attached by the same originating task.

scan table: The section of the PVT (page vector table) that contains the headers for the PCB queues.

SCB: STA control block.

SCT: Step control table.

SDWA: STA diagnostic work area.

second exit: A mechanism of alerting a routine that its request that a page be fixed and/or loaded has completed asynchronously.

secondary paging device: An auxiliary storage device that is not used for paging operations until the available space on primary paging devices falls below a specified minimum. Portions of a secondary paging device can be used for purposes other than paging operations.

segment: A continuous 64K area of virtual storage that is allocated to a job step or system task. Contrast with overlay segment.

segment table (SGT): A table used in dynamic address translation to control user access to virtual storage segments. Each entry indicates the length, location, and availability of a corresponding page table.

segment table entry (STE): An entry in the segment table that indicates the length, location, and availability of a corresponding page table.

segment translation exception: A program interruption that occurs when a virtual address cannot be translated by the hardware because the invalid bit in the segment table entry for that address is set. Synonymous with program interruption code 16. See also page translation exception, translation specification exception.

SGT: Segment table.

short-fixed: Pertaining to a request to fix one or more pages for a relatively short time.

SIOT: Step input/output table.

SIRB: System interruption request block.

SLIH: Second-level interruption handler.

slot: A continuous area on a paging device in which a page can be stored.

slot group: A set of slots on one or more tracks within a cylinder on a paging device.

slot number: A part of an external page address that refers to a slot; together with a device number and a group number, it identifies the location of a page in external page storage.

slot queue: A list of elements in the system nucleus that define the page slots available on one paging device. A slot queue is built for each paging device.

SMF: System management facilities.

snapshot dump: A selective dynamic dump performed at various points in a program.

SOD: Swap-out device work table.

space record A record that separates pages in a page data set.

SPCA: Swap communications area.

SPCT: Swap control table.

SPQE: Subpool queue element.

SQA: System queue area.

SQE: Supervisor queue element.

STA control block (SCB): A control block that contains information to be used by the ASIR routine for scheduling a user's STA exit routine during abnormal termination of a task.

STA diagnostic work area (SDWA): A work area that contains information about an abnormally terminating task. The ASIR routine uses the SDWA to schedule user-written diagnostic and retry routines.

stack: A list so constructed and maintained that the list element available for retrieval is the one that was most recently stored in the list. (Also called a last-in, first-out (LIFO) list, or pushdown list.)

STAR: System Task ABEND Recovery routine.

STE: Segment table entry.

step must-complete status: The status established by the control program that signifies that a designated task is to be the only dispatchable task within a job step. (Note that the initiator is also made nondispatchable.)

subpool: All of the storage blocks allocated under a subpool number for a particular task.

subpool queue element (SPQE): A VSS control block that describes allocated areas of storage assigned to a subpool.

subsystem: A secondary or subordinate system, usually capable of operating independently of or asynchronously with a controlling system.

subtask: A task that is initiated (attached) and terminated (detached) by a higher-level task.

subtask queue for a job step: The chain of tasks that are associated with a particular job step, showing the order in which the tasks were initiated (attached). See also TCB queues.

supervisor call (SVC) instruction: An instruction that interrupts the program being executed and passes control to the supervisor so that it can perform a specific service indicated by the instruction.

supervisor initialization: A general term referring to the execution of the initial program loader and the nucleus initialization program in loading and preparing the control program for operation.

supervisor lock: An indicator used to prevent entry to disabled code while a disabled page fault is being resolved. The paging supervisor is not restricted by this lock.

supervisor queue element (SQE): A control element used to schedule an asynchronous supervisor service.

supervisor request block (SVRB): A request block that represents a type-2, type-3, or type-4 SVC routine that must be executed on behalf of a task.

supervisor state: A state during which the central processing unit can execute input/output and other privileged instructions. Contrast with problem state.

SVC fix delay queue: In paging supervision, a queue on which fix requests (received through SVC entries to the FIX/LOAD routine) are placed when they cannot be processed because the amount of fixable real storage has fallen below a specified minimum.

SVC routine: A control program routine that performs or begins a control program service specified by a supervisor call instruction.

SVC table (SVCTABLE): A table that describes all SVC routines included in the system at system generation.

SVRB: Supervisor request block.

SWA: System work area.

SWAB: System work area block.

SWAH: System work area header.

swap communications area (SPCA): A control block that contains information necessary to effect a swap-out operation and complete a swap-in operation.

swap control table (SPCT): A control block that defines and reflects the status of a swap-in or swap-out operation.

swap-out device work table (SOD): A 32-byte internal work area used by paging supervision routines during swap-out processing

swap queue: In paging supervision, a queue on which the Swap Interface routine places PCBs representing swap requests (SVC 115).

swapping: In VS2 with TSO, a paging technique that writes the active pages of a job to external page storage and reads pages of another job from external page storage into real storage.

synchronous: Occurring with a regular or predictable time relationship. Contrast with asynchronous.

system generation: The process of using an operating system to assemble and link together all of the parts that constitute another operating system.

system interruption request block (SIRB): A single request block in the system, used by the system error task to represent an error-handling routine.

system management facilities (SMF): An optional control program feature that provides the means for gathering and recording information that can be used to evaluate system usage.

system must-complete status: The status established by the control program that signifies that a designated task is to be the only dispatchable task in the system (except for permanent system tasks which remain dispatchable). (The permanent system tasks are the paging task, the recovery management support task, the system error task, the dynamic support system task, the communications task, and the master scheduler task.)

system nucleus: That portion of the supervisor that resides in fixed frames in the lowest portion of real storage.

system queue area (SQA): An area of virtual storage reserved for system-related control blocks.

system queue origin list: See GOVRFLB.

system resource: Any facility of the computing system that may be allocated to a task.

system work area (SWA): One or more segments in virtual storage used as a work area for system programs.

system work area block (SWAB): A VSS control block that represents a SWA segment.

system work area header (SWAH): A VSS control block that begins a chain of system work area control blocks.

target device: A general term for the device from which the page slot is to be assigned for a page-out operation.

task control block (TCB): The consolidation of control information related to a task.

task interruption request block (TIRB): A request block (one per task) that is used to represent a control program routine that must be performed asynchronously when synchronous execution is impossible.

task level: A term that describes the position of one or more same-level tasks on a subtask queue in relation to other tasks on the queue. For the task(s) at one level, the originating task and all tasks above that task are called higher-level tasks or antecedent tasks; all tasks below it are called lower-level tasks or descendant tasks.

task post queue: In paging supervision, a queue on which a PCB is placed whenever a page is reclaimed in the process of a page-out or is no longer needed during a page-in. The PCB is placed on the queue to await processing that will determine whether additional paging services must be performed.

task queue: A queue of all the task control blocks (TCBs) present in the system at any time, chained in order of dispatching priority. See also TCB queues.

task switching: Saving the program status and task context of a task, and establishing the program status and task context of another task to effect the exchange of control to the second task.

TASK timing option: An option in the STIMER macro instruction. The option specifies that the time interval is to be decremented only when the associated task is active. Contrast with REAL timing option. See also WAIT timing option.

TCB: Task control block.

TCB queues: The two TCB queues are the subtask queue for the job step and the task queue. These queues consist of the same TCBs. On the subtask queue, the TCBs are in the order in which they were created for the job step. The ABEND routine uses this queue to establish the order in which a job step's resources are freed. On the task queue, all TCBs are in order of dispatching priority. The dispatcher scans down this queue to determine the TCB for the highest priority ready task. See also subtask queue for a job step and task queue.

thrashing: A condition in which the system can do little useful work because of excessive paging.

time-of-day (TOD) clock: A System/370 CPU feature used to measure elapsed time in microseconds. It is a 64-bit incrementing binary counter. A one is added to bit position 51 every microsecond. Timer supervision uses the time-of-day clock to calculate the time of day and maintain the current date.

time slice control element (TSCE): A control element used to control the dispatching of tasks in a time slice group.

timer queue element (TQE): The control block used by timer supervision to record the information necessary to schedule and process a request for a timed interval.

timer units: A representation of a time interval request in which the least significant bit is the equivalent of 26.04166 microseconds.

TIRB: Task interruption request block.

TOD clock: Time-of-day clock.

top terminating task: A task that is being abnormally terminated and is the highest-level task in a set of subtasks that must be terminated. The full set of subtasks that must be terminated is called the chain of terminating tasks.

TQE: Timer queue element.

trace: (1) The record of a series of events. (2) To record a series of events as they occur.

trace table: One in a chain of tables containing a history of a task's environment during processing.

translation specification exception: A program interruption that occurs when a page table entry, segment table entry, or the control register pointing to the segment table contains information in an invalid format. Synonymous with program interruption code 18. See also page translation exception, segment translation exception.

TSCE: Time slice control element.

translation tables: Page tables and segment tables.

type-1 SVC message table: The table containing entries, supplied by a type-1 SVC routine, that indicates failures in the type-1 SVC routine.

validity map: A control block that defines the storage addresses that can be referred to by a non-key-0 program in a virtual storage region.

virtual address: An address that refers to virtual storage and must, therefore, be translated into a real storage address when it is used.

virtual block number: The virtual address (that is, the segment number and page number) associated with a page.

virtual equals real (V=R) storage: An area of virtual storage that has the same range of addresses as real storage and is used for a program or part of a program that cannot be paged during execution. Preferred term is nonpageable dynamic area.

virtual equals virtual (V=V) storage: An area of virtual storage used for a program or part of a program that can be paged during execution. Preferred term is pageable dynamic area.

virtual storage: Addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, rather than by the actual number of real storage locations.

virtual storage region: A subdivision of the dynamic area that is allocated to a job step or system task.

virtual subarea list (VSL): A control block describing an area of virtual storage on which some paging supervisor operation is to be performed.

V=R dynamic area: Preferred term is nonpageable dynamic area.

V=R intercepted page: A page frame below the V=R line that is currently being used for a purpose other than as part of a V=R region and is now needed for allocation to a new V=R region. The page frame table entry (PFTE) for the frame is marked so that, when the frame is released from its current use, it will be assigned to the new V=R region rather than being added to the available page queue.

V=R line: The high address limit of the nonpageable dynamic area.

V=R region: Preferred term is nonpageable region.

VSL: Virtual subarea list.

VSS: Virtual storage supervision.

V=R root PCB queue: A queue of root PCBs awaiting deferred allocation.

V=V dynamic area: Preferred term is pageable dynamic area.

V=V region: Preferred term is pageable region.

WAIT timing option: An option of the STIMER macro instruction. The option specifies that the requested interval is to be decremented continuously, and that the associated task is to be placed in the wait condition until the interval is completed.

working set: The set of a user's pages that must be active in order to avoid excessive paging.

XPT: External page table.

XPTE: External page table entry.

XTLST: Extent list.

Index

Indexes to program logic manuals are consolidated in the publication OS/VS Master Index of Logic, GY28-0603. For additional information about any subject listed on the following pages, refer to other publications listed for that subject in the master index.

Index

Where more than one page reference is given, the major reference is first.

SY27-7244-1

OS/VS2 Supervisor Logic (Volume 1) Printed in U.S.A. SY27-7244-1

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)

OS/VS2 Supervisor Logic
(Volume 1)

SY27-7244-1

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.*

Please check or fill in the items below, adding explanations and other comments in the space provided.

How did you use this publication?

☐ As an introduction ☐ As a reference manual ☐ As a text (student) ☐ As a text (instructor)

☐ For another purpose (explain)____

____

	Very Useful	*Sometimes Useful*	*Unnecessary*	*Other*
General Organization	☐	☐	☐	____________
Overall Introduction	☐	☐	☐	____________
Section Introductions	☐	☐	☐	____________
Program Organization Diagrams	☐	☐	☐	____________
Routine Synopses	☐	☐	☐	____________
Data Area Maps	☐	☐	☐	____________
Registers on Entry and Exit	☐	☐	☐	____________
Control Blocks Matrix	☐	☐	☐	____________

Do the method of operation diagrams and the notes contain:

DIAGRAMS	NOTES
☐ Too many details	☐ Too many details
☐ Not enough details	☐ Not enough details
☐ Right amount of details	☐ Right amount of details

Check any of the following that you would omit from the method of operation diagrams:

☐ Subroutine boxes
☐ Entry point labels
☐ Data reference arrows
☐ Data modification arrows
☐ Data movement arrows

Other comments, criticisms, errors, etc. Please be specific where possible.

What is your occupation?____

Number of latest Technical Newsletter (if any) concerning this publication:____________________

Please include 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.)

Cut or Fold Along Line

SY27-7244-1

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems 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 40
Armonk
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 636
Neighborhood Road
Kingston, New York 12401

Fold   Fold

OS/VS2 Supervisor Logic (Volume 1) Printed in U.S.A. SY27-7244-1

IBM®