LY28-1189-2
File No. S370-37

Program Product

MVS/Extended Architecture Service Aids Logic

MVS/System Product
JES3 Version 2 5665-291
MVS/System Product
JES2 Version 2 5740-XC6

IBM

Third Edition (July, 1985)

This is a major revision of, and obsoletes, LY28-1189-1. See the Summary of Amendments following the Contents for a summary of changes to this manual.

This edition applies to Version 2 Release 1.3 of MVS/System Product JES2 - 5740-XC6 and Version 2 Release 1.2 of MVS/System Product JES3 - 5665-291 until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

Preface

This publication describes the internal logic and organization of the five service aid programs provided for use in servicing MVS/XA. The publication is intended for the IBM programming systems representative who is involved in maintaining the service aid programs. For information about the use and operation of the service aid programs, refer to *System Programming Library: Service Aids*, GC28-1159.

How This Publication is Organized

This publication contains five chapters preceded by a General Information section and an Abbreviation Dictionary and followed by an Index. Each of the chapters corresponds to one of the service aid programs. Notice that the chapters are arranged in alphabetical order by the shortened service aid program name.

The General Information section introduces the concept of a service aid briefly describes each of the service aid programs. The Abbreviation Dictionary lists some of the abbreviations and acronyms used in this publication and their meanings.

Each chapter is divided into the following sections:

- **The Introduction** — a description of the service aid in general with some discussion of external characteristics.
- **Method of Operation** — a functional approach to the program using both diagrams and text.
- **Program Organization** — a description of program loading, storage layout, module calling sequences, and the modules themselves.
- **Data Areas** — a description of the major data areas used by the service aid program. Where applicable, this section contains references to detailed descriptions of the data areas in other publications.
- **Diagnostic Aids** — information that can be useful for diagnosing problems in the service aid program.

Related Publications

The following publications are referred to in the text:

MVS/Extended Architecture System Programming Library: Service Aids, GC28-1159.

MVS/Extended Architecture SYS1.LOGREC Error Recording, GC28-1162.

MVS/Extended Architecture Debugging Handbook.

Volume 1 — LC28-1164
Volume 2 — LC28-1165
Volume 3 — LC28-1166
Volume 4 — LC28-1167
Volume 5 — LC28-1168

MVS/Extended Architecture Interactive Problem Control System Logic and Diagnosis, LY28-1298

MVS/Extended Architecture Message Library: System Messages, GC28-1156.

MVS/Extended Architecture System Logic Library (multiple volumes). Volume 1, LY28-1208, contains order numbers for all volumes.

ACF/VTAM Version 2 Diagnosis Guide, SY38-0615.

Contents

Figures

General Information

A service aid program is intended to aid in the diagnosis of system and application program failures. The main functions of a service aid are to:

- Collect data that relates to the failure.
- Format and print the data in a form applicable to debugging.
- Aid in developing and applying an immediate fix for a problem.

The following programs are service aids:

- The generalized trace facility (GTF), which traces selected system and application program events and records the data for formatting and printing by the AMDPRDMP service aid program or by the ABEND/SNAP routines.

- AMASPZAP, which inspects and modifies data in a load module that is part of a partitioned data set or in a specific data record that is contained in a direct access data set; AMASPZAP also dumps records from data sets residing on direct access storage devices.

- AMBLIST, which produces formatted object module, load module, nucleus, and cross-reference listings, as well as load module and link pack area maps and CSECT identification record information.

- AMDPRDMP, which formats and prints the contents of the AMDSADMP output data set, the SYS1.DUMPnn data sets, a SYSMDUMP ABEND dump, or any dumps produced by SVC dump, and the GTF trace data set.

- AMDSADMP, which produces a dump of real storage, instruction trace data (created by the console-initiated loop recording) and critical areas of each active address space.

The following service aid is described in the publication *MVS/Extended Architecture SYS1.LOGREC Error Recording Logic*.

- IFCDIP00, which initializes the SYS1.LOGREC data set.

The following service aid is described in *Environmental Recording Editing and Printing (EREP) Program Logic*.

- IFCEREP1, which edits records from the SYS1.LOGREC data set and writes them to a specified output device.

Abbreviation Dictionary

Abbreviation	Meaning
AID	record identifier
ARB	address range block
ASCB	address space control block
ASID	address space identifier
ASXB	address space control block extension
ASVT	address space vector table
ASMVT	auxiliary storage manager vector table
BCB	buffer control block
BCT	buffer control table
BSAM	basic sequential access method
CCT	common communication/control table
CCW	channel command word
CDE	contents directory entry
CESD	composite external symbol dictionary
CHPID	channel path identifier
CS	control section name
CSCB	command scheduling control block
CSCH	clear subchannel
CSD	common system data area
CSECT	control section
CVT	communication vector table
DA	data area or direct access
DAT	dynamic address translation
DCB	data control block
DEB	data extent block
DLIB	distribution library
DQE	description queue element
DS	data set
DSCB	data set control block
DSCE	dynamic storage control element
EBCDIC	extended binary-coded-decimal interchange code
ECB	event control block
ECT	exit control table
ECTE	exit control table entry
EID	event identifier
EOF	end of file
EOV	end of volume
EP	entry point name
EPA	entry point address
ERB	error recovery block
ESD	external symbol dictionary
FID	format identifier

FXTAB	fix table
GSMQ	global service manager queue
GSPL	global service priority list queue
GTF	generalized trace facility service aid program
GTFBCB	GTF buffer control block
GTFBLOK	GTF blocking area
GTFBUFR	GTF buffer
GTFPCT	GTF primary control table
HSCH	halt subchannel
ICR	independent component release
IDR	CSECT identification record
INITDATA	initialization data
I/O	input/output
IODB	I/O device block
IOS	input/output supervisor
IPL	initial program load
IQE	interruption queue element
IRB	interruption request block
JCL	job control language
JFCB	job file control block
JOBNAME	job name
LCCA	logical configuration communication area
LCCAVT	logical configuration communication area vector table
LGVT	logical group vector table
LIST	AMBLIST service aid program
LLE	load list element
LPA	link pack area
LPID	logical page identifier
LPRB	loaded program request block
LR	label reference
LRECL	logical record length
LSID	logical slot identifier
LSMQ	logical service manager queue
LSPL	logical service priority list
LSQA	local system queue area
LT	logical track
LTH	logical track header
MC	monitor call
MCAWSA	monitor call application work/save area
MCCD	monitor call class directory
MCCE	monitor call control element
MCCLE	monitor call class element
MCED	monitor call event directory
MCEE	monitor call event element
MCHEAD	monitor call base table
MCQE	monitor call queue element
MCRWSA	monitor call router work/save area
MN	module name
MSCH	modify subchannel
PCB	print control block
PCI	program controlled interruption
PDS	partitioned data set
PER	program event recording
PFT	page frame table

PICA	program interruption control area
PRDMP	AMDPRDMP service aid program
PSW	program status word
PTF	program temporary fix
QCB	queue control block
QCR	queue control record
QEL	queue element
RAD	RSM address space data
RANGETAB	range table
RB	request block
RCB	recovery control block
RCSW	real channel status word
RE	record entry
RECFM	record format
RLD	relocatable load dictionary
RLT	relocation table
RNIO	remote network input/output
RQE	replay queue element
RSM	real storage management
SADMP	AMDSADMP service aid program
SAT	swap address table
SCSW	subchannel status word
SD	section definition
SDATA	service data area
SLE	save list element
SLH	subchannel logout handler
SLIP	serviceability level indication processing
SPZAP	AMASPZAP service aid program
SQA	system queue area
SR	subroutine
SSCH	start subchannel
SSI	system index status
STA	starting address
SUT	storage use table
SVC	supervisor call
SYSGEN	system generation
SYSIN	system input
SYSOUT	system output
TCAM	telecommunications access method
TCB	task control block
TIOT	task input/output table
TOD	time of day
TQE	timer queue element
TTR	relative trace and record address
UCB	unit control block
VCCT	virtual common communications table
VOLID	volume identification
VPA	virtual page address
VS	virtual storage

Summary of Amendments

**Summary of Amendments
for LY28-1189-2
As Updated July 12, 1985**

This revision contains new and updated material in support of MVS/XA System Product Version 2, release 1.3, and includes the following:

- For AMDPRDMP:
  - several new AMDPRDMP control statements: JES3, SADMPMSG, and TCAMMAP.
  - diagrams of the Verb Exit Processing section are now in table form
  - minor technical and editorial revisions
- For AMDSADMP:
  - the addition of three new modules:
    - AMDSAXSM
    - AMDSAFCM
    - AMDSADCM
- numerous changes to HIPOs and extended descriptions
- minor technical and editorial revisions

**Summary of Amendments
for LY28-1189-1
as Updated January 31, 1984**

This is a major revision of LY28-1189-0. It contains new and updated information in support of MVS/XA System Product and includes the following:

- For AMDPRDMP:
  - support for the ForMAT and SUMMARY control statements have changed
- For AMDSADMP:
  - support for the Magnetic Tape Subsystem Display
  - support for the 3290 console
- minor technical and editorial changes

**Summary of Amendments
for LY28-1189-0
as Updated March 31, 1983
by TNL LN28-0786**

This technical newsletter contains new and updated material in support of MVS/XA System Product and includes the following:

- A revised description of monitor call event handling in GTF.
- A description of modifications to SADMP that support the 3350 DASD with a 3380-11 control unit.
- Two new control statements in AMDPRDMP: MTRACE and DAEDATA.
- Minor technical and editorial changes.

Chapter 1. Generalized Trace Facility (GTF)

Section 1: Introduction

The Generalized Trace Facility (GTF) traces selected system and application program events and records the data for later formatting and printing by the AMDPRDMP service aid program. GTF functions independently of the system trace facility. If system trace is active when GTF is started, system trace remains active. Starting GTF does not alter the status of system trace.

Invoking and Controlling GTF

The operator invokes GTF as a system task with a START command specifying the cataloged procedure. (This procedure has the membername of GTF and is included in SYS1.PROCLIB at system generation.) The operator controls GTF processing via parameters in the START command and the TRACE options. The information supplied in the START command includes the following:

- MODE — Defines whether the trace data is recorded in the GTF address space (internally) or in an external data set defined by the IEFRDER DD statement in the cataloged procedure.
- TIME — Provides the option for every logical trace record to be time-stamped with the TOD clock value at the time the record is placed into the buffer.
- DEBUG — Permits GTF to either attempt recovery or terminate from errors it encounters while building a trace record.
- BUF — Specifies the number of trace buffers which GTF makes available for the ABDUMP/SNAP, SVC dump, or AMDSADMP program to include in dump output.
- The operator may also specify parameters to override JCL parameters in the cataloged procedure or values for symbolic parameters in the cataloged procedure.

For a description of the START command, its GTF parameters, and the GTF cataloged procedure, refer to *Service Aids*.

GTF Trace Options

GTF obtains the trace options from the operator, or, if a SYSLIB DD statement is provided in the cataloged procedure, from the specific member of SYS1.PARMLIB. The following options may be specified.

- ASIDP — requests that GTF tracing be limited to specific address spaces (which the user is prompted to supply).
- CCW — requests recording of channel programs and associated data for start subchannel and resume subchannel operation and I/O interruptions.
- CCWP — requests recording of channel programs using specific channel program trace options (which the user is prompted to supply) for start subchannel and resume subchannel operations or I/O interruptions or both.
- CSCH — requests recording of data for clear subchannel operations.
- DSP — requests recording of comprehensive data (or minimal data if SYSM is also specified) for all task, LSR, or SRB dispatch events.
- EXT — requests recording of comprehensive data for all external interruptions.
- HSCH — requests recording of data for halt subchannel operations.
- IO — requests recording of data for all I/O interruptions except program-controlled interruptions (PCI's).
- IOP — requests recording of data for I/O interruptions on specific devices (which the user is prompted to supply).
- JOBNAMEP — requests that GTF tracing be limited to specific jobs (which the user is prompted to supply).
- MSCH — requests recording of data for modify subchannel operations.
- PCI — requests recording of data for all intermediate status interruptions. If IOP or SYSP is also specified, only intermediate status interruptions on the specified devices will be recorded.
- PI — requests recording of comprehensive data for all program interruptions.
- PIP — requests recording of comprehensive data for program interruptions with specific interruption codes (which the user is prompted to supply).
- RNIO — requests recording of comprehensive data (or minimal data if SYSM is also specified) for all basic transmission units (BTU's) received by ACF/VTAM.
- RR — requests recording of comprehensive data (or minimal data if SYSM is also specified) for all entries to functional error recovery routines such as STAE/ESTAE routines.

- SIO, SIOP — If you request the SIO or SIOP option, GTF processes your request as a request for SSCH or SSCHP.
- SLIP — requests recording of data specified by a SLIP trap with a tracing action when it matches or when any trap is checked with SLIP DEBUG mode specified.
- SRM — requests recording of comprehensive data (or minimal data if SYSM is also specified) for all entries to the system resources manager.
- SSCH — requests recording of data for all start subchannel and resume subchannel operations.
- SSCHP — requests recording of data for start subchannel and resume subchannel operations on specific devices (which the user is prompted to supply).
- SVC — requests recording of comprehensive data for all SVC interruptions.
- SVCP — requests recording of comprehensive data for specific SVC numbers (which the user is prompted to supply).
- SYS — requests recording of comprehensive data for all external interruptions, program interruptions, recovery routines, and supervisor call interruptions. SYS causes recording of all I/O interruptions, start subchannel and resume subchannel operations, clear subchannel operations, halt subchannel operations, and modify subchannel operations.
- SYSM — requests recording of minimal data for the events listed under SYS.
- SYSP — requests recording of comprehensive data for specific system events (which the user is prompted to supply).
- TRC — requests recording of data for all events associated with the trace task itself.
- USR — requests that all data passed to GTF via the GTRACE macro instruction be recorded in the trace data set.
- USRP — causes GTF to build an internal table of the user event identifiers (EIDs) that the user specifies. The TEST parameter of the GTRACE macro tests whether or not tracing is active for the specified EIDs.

GTF Hooks

After the operator starts GTF and selects the trace options, GTF waits for an event that requires GTF tracing, GTF is notified of such an event by a hook from another routine. Hooks are issued by IBM users, IBM components, first level interruption handlers, the I/O supervisor, the dispatcher, the recovery/termination manager, and the system resources manager. GTF also receives hooks to indicate that trace data is to be included in a dump; these hooks are issued by ABEND or SVC DUMP.

Two macro instructions provide the hooks into GTF:

- GTRACE — which is a trace hook issued by users and IBM components to cause GTF to write their data in the GTF output. Events traced by the GTRACE macro will use an Event Identifier (EID) from one of the three ranges listed below:

  1. 0000 — 1023 user events
  2. 1024 — 1535 reserved for program products
  3. 1536 — 4095 reserved for IBM components and subsystems

  EIDS in the first range are available for general use by all GTF users. EIDS in the second and third ranges are reserved.

- HOOK — which is issued by system routines to notify GTF of an event to be traced or of the need for trace data for a dump.

Both of these macro instructions generate the monitor call (MC) instruction (see Section 6, a maskable interruption).

GTF Output

GTF builds two kinds of records:

- Trace records — which contain information about system events. If the SYSM (system minimal) option is in effect, the trace record is similar to that built by the system trace facility. If the trace option is other than SYSM, the trace record contains more comprehensive data about the system event.

- Control records — timestamp and lost block/event records. The timestamp record is the first record in every block; it identifies the time of day and date when the first trace entry was placed in the block. The lost block/event record contains information about events that were missed.

When trace mode is internal, the records are maintained in 4096-byte buffers in variable blocked format. When mode is external, the trace records are written in the external data set (either tape or direct access) defined by the IEFRDER DD statement in the cataloged procedure. When the end of the data set on a direct access device is reached, writing continues at the beginning of the data set. The data will be in variable blocked format with a blocksize of 4096 bytes.

Formatting and Printing GTF Output

The AMDPRDMP service aid program formats the GTF output for printing. The EDIT control statement in AMDPRDMP provides:

- Data reduction for GTF output.

- Selectivity of data from the trace output data set.

- An interface to user- or component-supplied formatting routines for data recorded via the GTRACE macro instruction.

For a detailed description of the EDIT function of AMDPRDMP, refer to the "Print Dump (AMDPRDMP)" chapter in this publication.

While GTF is active and SDATA=TRT is in effect, the ABEND/SNAP routine formats and prints trace records related to the dumped address space. The records formatted are those from the n most recent GTF buffers, where n is the number specified with the BUF=keyword on the START command.

Section 2: Method of Operation

This section describes how GTF collects and manages data about system events. As shown in GTF Diagrams 1 and 2, the description begins with an overview of GTF, then describes the stages of the overall processing listed below. Finally, it describes the processing of the monitor call-routing facility.

- GTF initialization.
- Hook processing.
- Trace record buffering and blocking.
- GTF-writer processing.
- GTF/dump interface.
- Error recovery.
- GTF termination.

Reading Method of Operation Diagrams

Method of operation diagrams are arranged in an input-processing-output layout: the left side of the diagram contains the data that serves as input to the processing steps in the center of the diagram, and the right side contains the data that is output from the processing steps. Each processing step is numbered; the number corresponds to the verbal description of the step in the extended description. While the processing step in the diagram is in general terms, the corresponding text is a specific description that includes a cross-reference to the code for the processing.

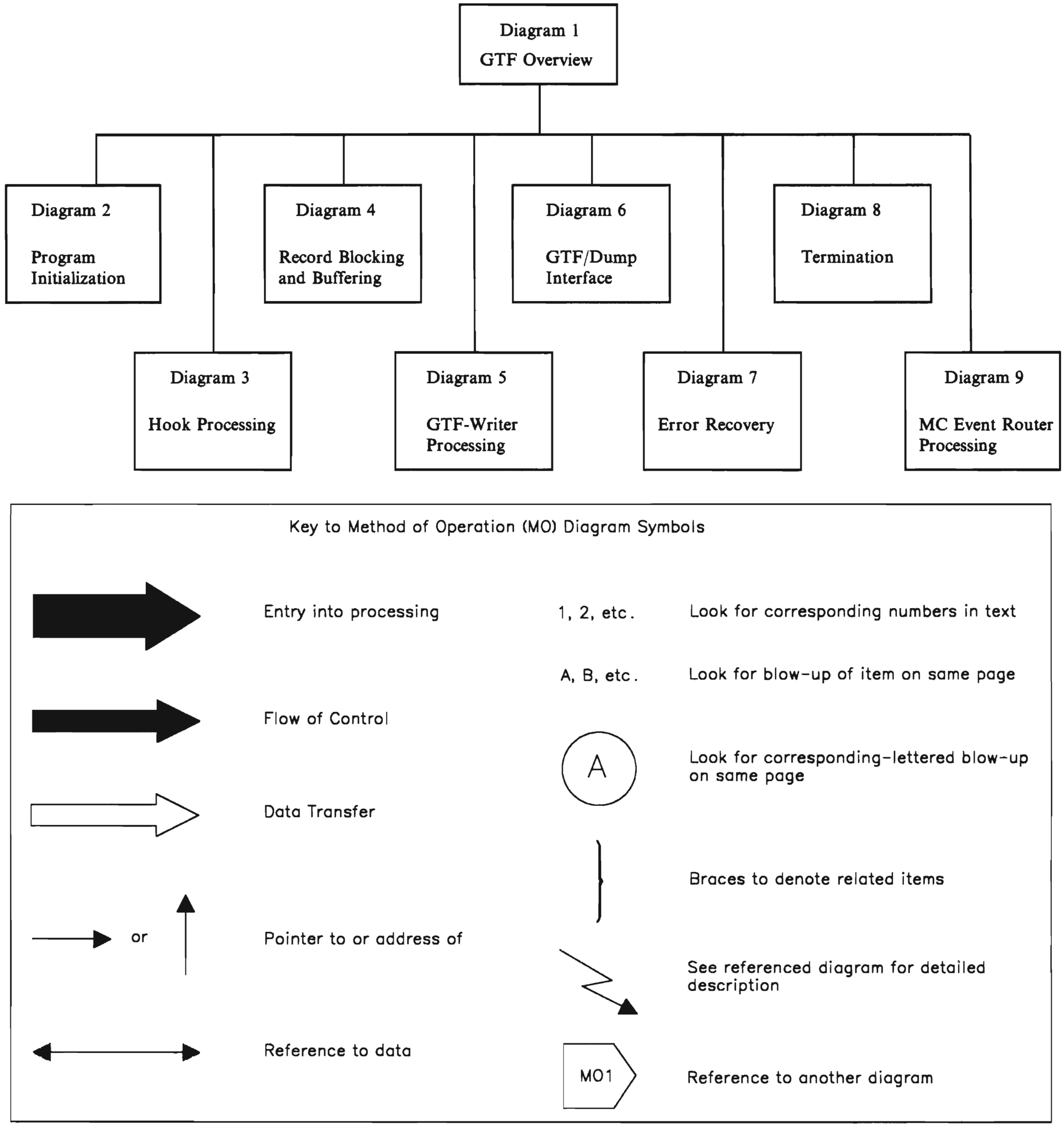
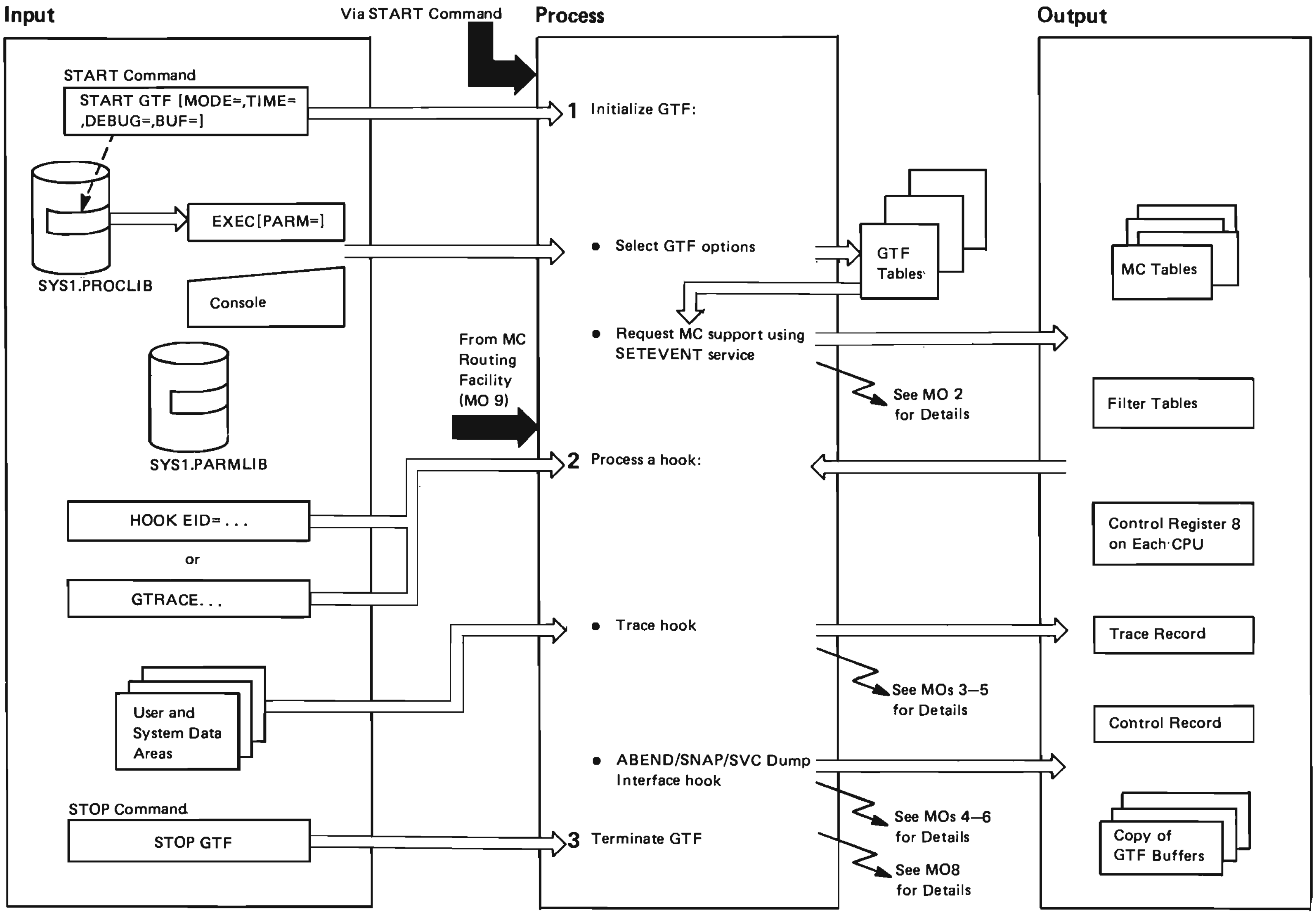


Figure 1-1. Key to Method of Operation Diagrams for GTF

GTF Diagram 1. GTF Overview (Part 1 of 2)



GTF Diagram 1. GTF Overview (Part 2 of 2)

Extended Description	Module	Label

1 The GTF procedure resides in SYS1.PROCLIB. The operator invokes GTF by use of a START command.

- Four parameters (MODE, TIME, DEBUG, and BUF) may be specified in the START GTF command or they may be specified in the EXEC statement in the GTF procedure. The options for the trace phase of GTF may be specified by the operator from the console or they may be included in a member of SYS1.PARMLIB. GTF obtains storage for tables in which it places information about the selected options.
- Before tracing can begin, GTF requires monitor call (MC) routing-facility support. The MC routing facility is an extension of the program first-level interruption handler (PFLIH). GTF issues the SETEVENT macro instruction to request MC support. The SETEVENT service constructs MC tables for the events GTF will trace and turns on the corresponding class bits in control register 8 for each active CPU. See Section 6 for a description of the SETEVENT macro instruction. If selective event tracing is in effect, GTF constructs filter tables.

2 When a program issues a HOOK or GTRACE macro instruction (see Appendix A) for an event to be traced, the MC routing facility passes control to GTF. GTF's processing depends on the kind of hook received:

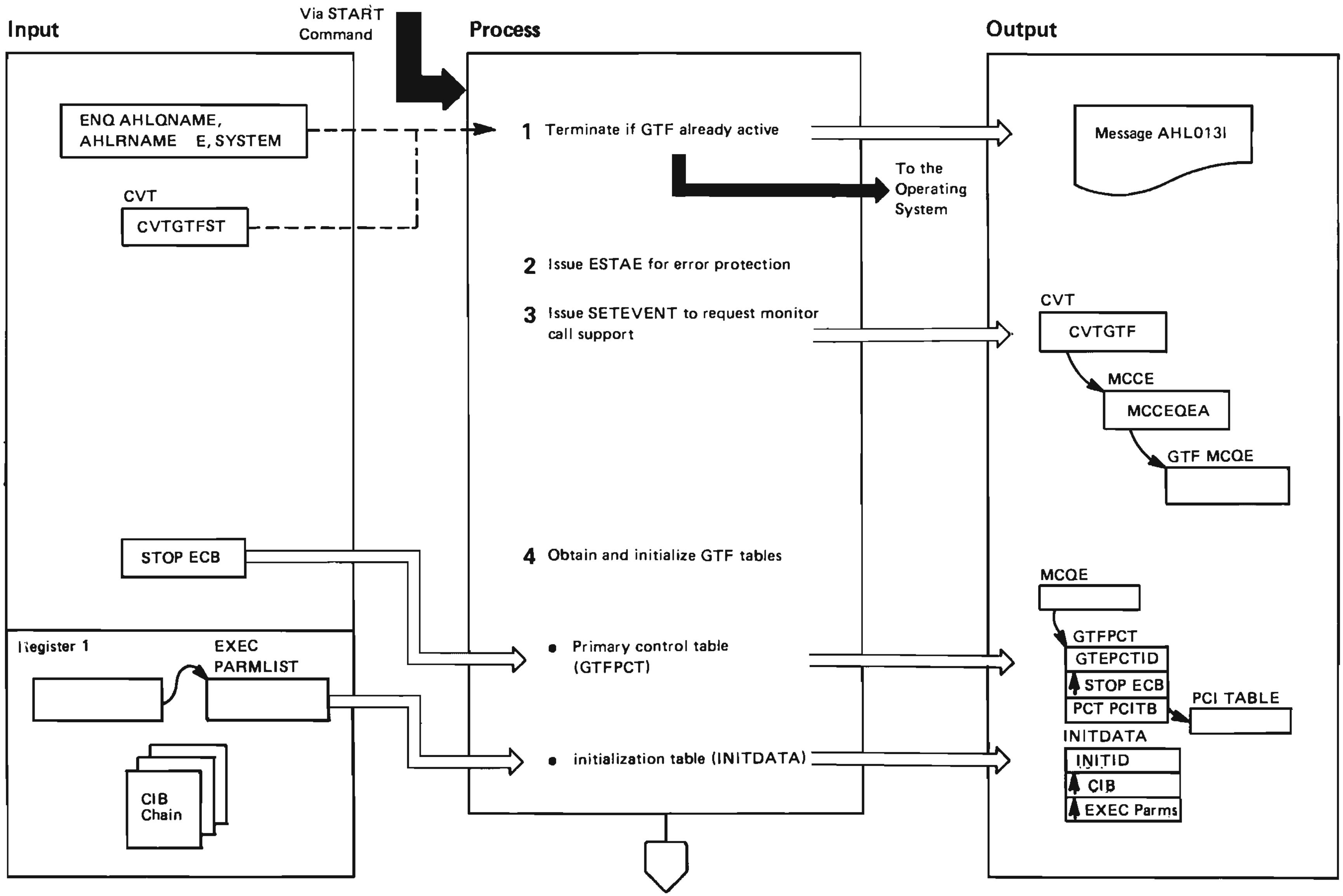
- Trace hooks notify GTF of an event that may be traced by GTF. GTF collects data about the event from user and/or system data areas, then records the data in trace records.
- Dump interface hooks from ABEND/SNAP/SVC Dump request trace data to be saved for a dump. GTF builds a control record and saves a copy of the current buffer contents. During dump processing, ABEND/SNAP/SVC Dump issues the AHLREAD macro instruction for GTF to pass the buffer copies to the dump program.

GTF records reside either in the GTF address space or in an external data set, depending on the MODE option in effect.

Extended Description	Module	Label

3 The operator terminates GTF via a STOP command. (See MO 8 for GTF termination. Abnormal termination is described in MO 7.)

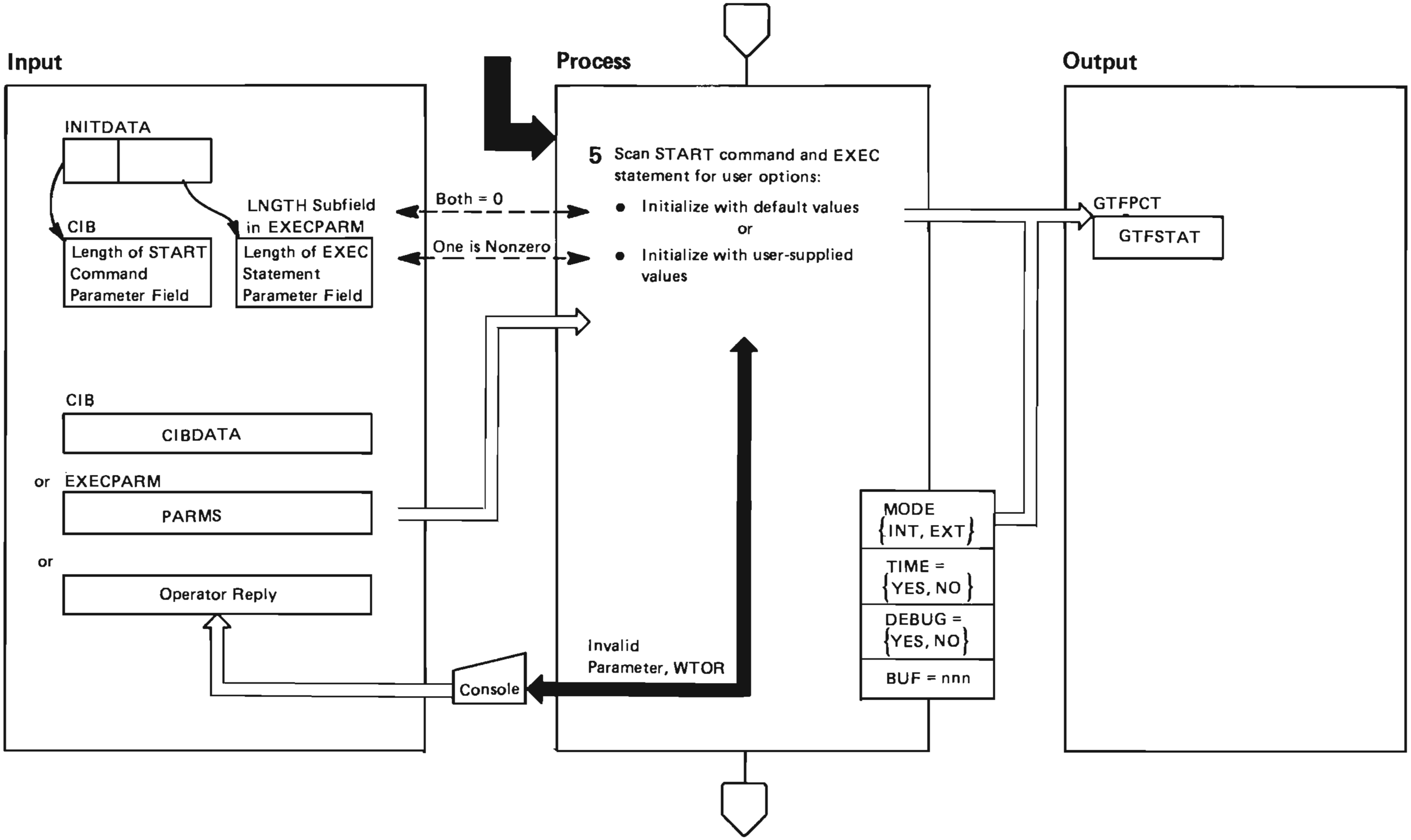
GTF Diagram 2. Program Initialization (Part 1 of 8)



GTF Diagram 2. Program Initialization (Part 2 of 8)

Extended Description	Module	Label
1 As soon as GTF receives control, it ensures that it is not already in operation from a previous START command. It issues an ENQ macro instruction for a major resource of VS2TRACE and a minor resource of GTF. If the resources are unavailable, or GTF initialization is in process or GTF is active, GTF issues message AHL013I to the console and returns control to the system. GTF then terminates.	AHLGTFI	
2 GTF provides error protection for initialization via an ESTAE macro instruction.	AHLGTFI	
3 GTF requests MC support via the SETEVENT macro instruction. As a result of SETEVENT processing, an MC queue element (MCQE) for GTF is established. It is chained off the MC control element (MCCE) from MCHEAD.	AHLGTFI SETEVENT service	
4 GTF obtains space in SQA for its primary control table, the GTFPCT. A flag is set in the GTF audit field (AUDITWRD), indicating that GTFPCT has been obtained. This field tracks GTF processing and determines what actions are taken if the GTF initialization ESTAE recovery routine receives control. GTF also obtains storage in subpool zero for the initialization data area INITDATA. If PCI and CCW or CCWP are requested, GTF builds the PCI table. It clears the tables and places EBCDIC identifiers in each table. GTF stores a pointer to the STOPECB in GTFPCT and stores pointers to the command input buffer (CIB) and the PARM field of the AHLGTF EXEC statement in INITDATA. If either area could not be obtained, GTF issues message AHL130I and returns control to the system.	AHLGTFI	

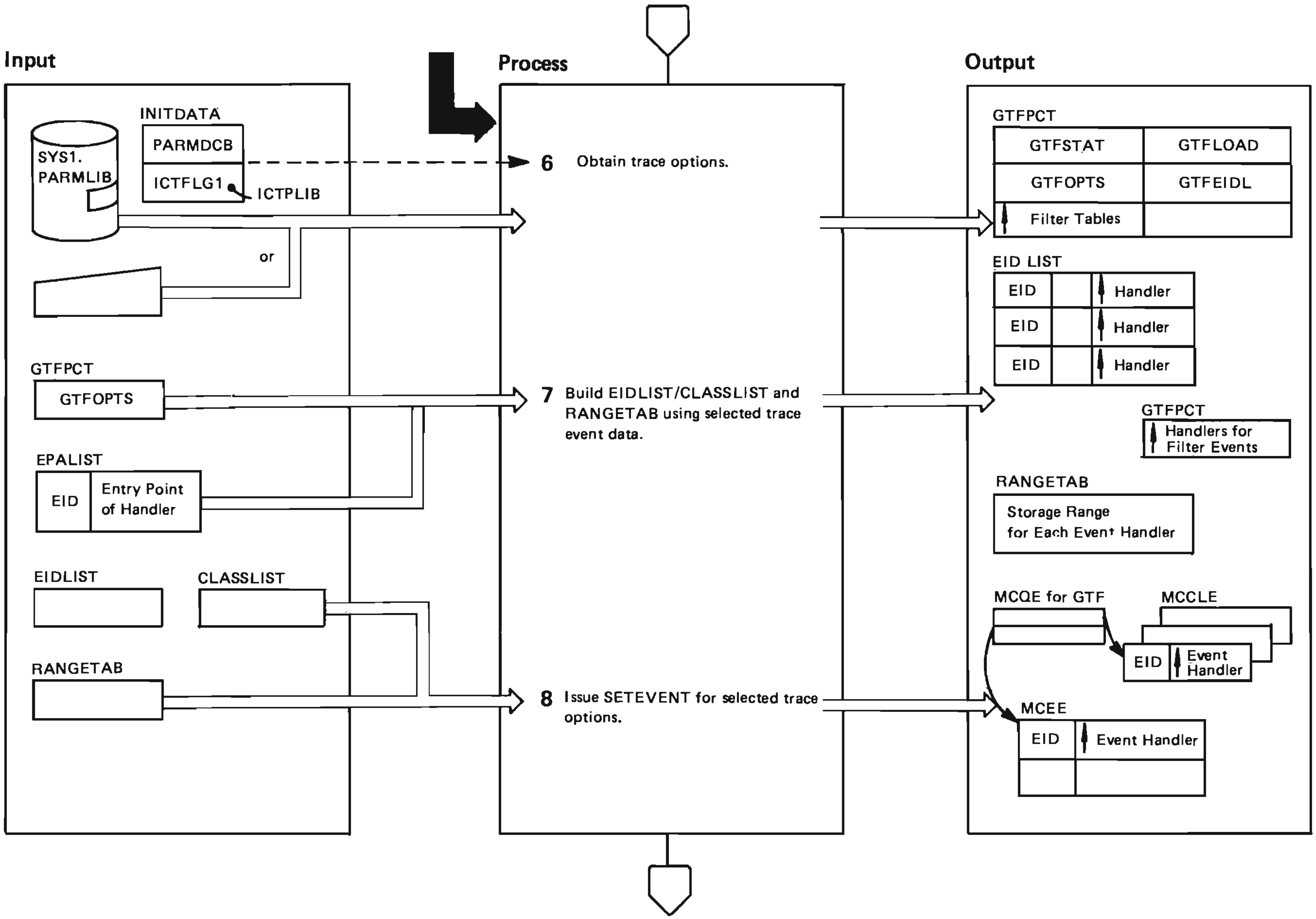
GTF Diagram 2. Program Initialization (Part 3 of 8)



GTF Diagram 2. Program Initialization (Part 4 of 8)

Extended Description	Module	Label
5 GTF checks the CIBDATLN field of the CIB to to determine whether there are parameters in the START command. If the field is nonzero, GTF obtains the parameters from the CIBDATA field and sets the appropriate option bits in the GTFSTAT field of the GTFPCT. If the CIBDATLN is zero, there are no parameters in the START command, so GTF checks the LNGTH subfield of the EXCPARM field in INITDATA. If it is nonzero, GTF obtains parameters from the EXEC statement and sets the appropriate bits in GTFSTAT. If an invalid parameter is encountered, GTF issues a WTOR for valid options. If parameters were not supplied via the START command or EXEC statement, the GTF defaults are in effect.	AHLSCAN	

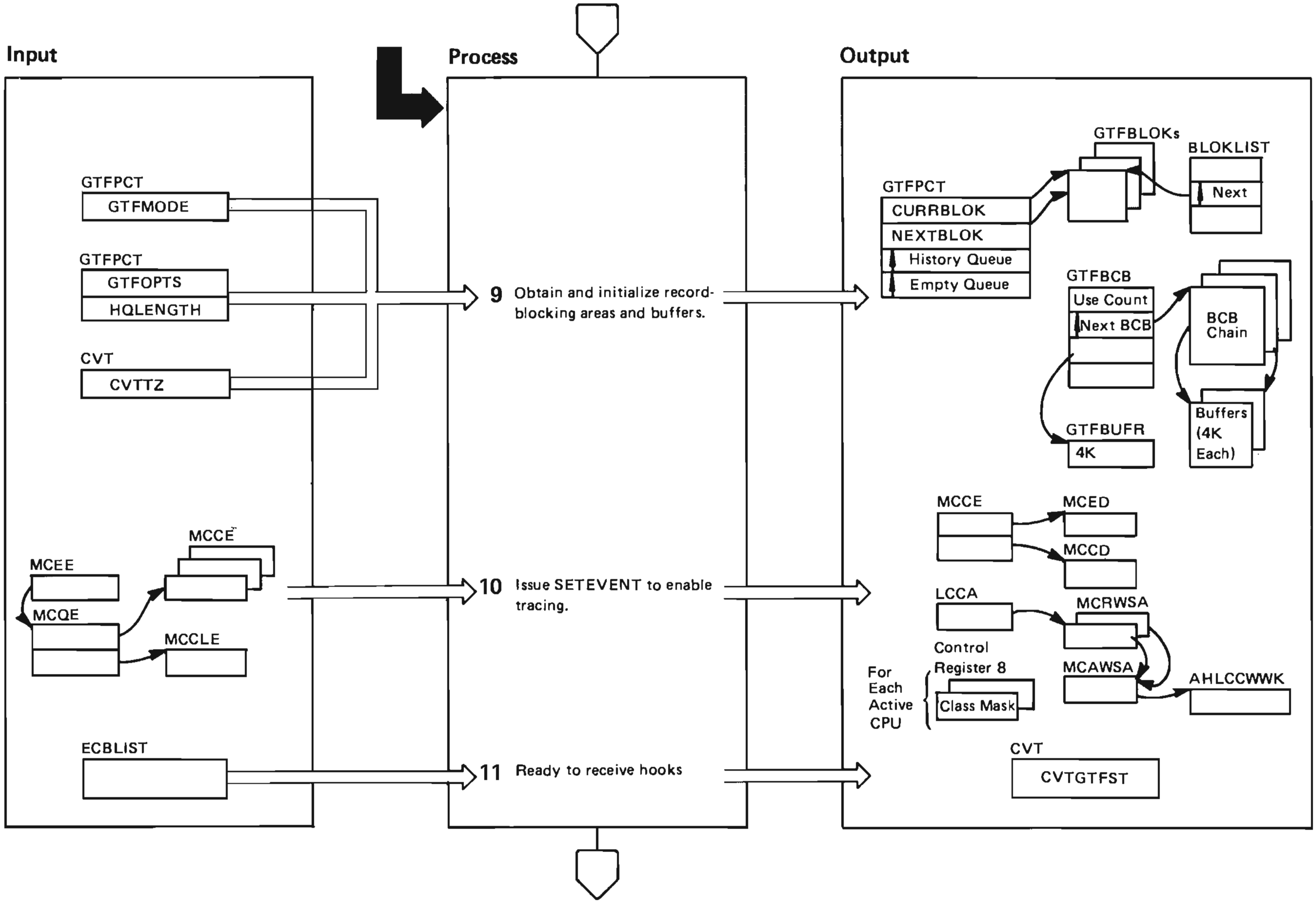
GTF Diagram 2. Program Initialization (Part 5 of 8)



GTF Diagram 2. Program Initialization (Part 6 of 8)

Extended Description	Module	Label
6 GTF determines the source of trace option input: either SYS1.PARMLIB or the console. A READJFCB is issued for the SYSLIB DD, and if PARMLIB input is indicated, the ICTPLIB flag in INITDATA is set to 1 and SYS1.PARMLIB input is read for the options. If input is to come from the console or if the SYS1.PARMLIB data set open is unsuccessful, the ICTPLIB flag is set to 0 and GTF issues AHL100A to obtain the options from the console. In either case, GTF obtains the options and sets the appropriate bits in GTFOPIND field of INITDATA. It also sets the PCTCATF bits in GTFPCT. GTF resolves specification of mutually exclusive options. (See Service Aids for a discussion of GTF options.) If the GTFOPIND bits indicate SVCP, SYSP, SSCHP, IOP, PIP, ASIDP, JOBNAMEP, or CCWP, GTF prompts the operator for selected trace events and specific trace options. After all options and filter events have been obtained, GTF issues message AHL103I. If selective tracing is in effect, GTF uses the EBCDIC tables to build filter tables or masks. The addresses of the filter tables are placed in GTFPCT except when ASIDP or JOBNAMEP are in effect. If ASIDP or JOBNAMEP is in effect, the filter tables for ASIDP or JOBNAMEP (not the addresses of the filter tables) are placed in the GTFPCT.	AHLCTL1 AHLTSCN AHLTPMT AHLTCTL AHLT103	
7 GTF determines which event handlers are required based on the options in effect as indicated by the GTFOPTS bits. Starting and ending addresses of the required modules are determined via the entry point list (EPALIST). GTF constructs an EIDLIST which contains EIDs of events to be traced and the entry points of the associated event handler modules. GTF also constructs the RANGETAB which contains starting and ending addresses of the event handler modules. From this table, it determines the range of storage required to fix GTF in storage. This range is set in FIXTAB which is passed as a parameter on the PGFIX macro instruction. The storage for FIXTAB is obtained from SQA since it is required to free storage of GTF termination.	AHLGTFI	(MCQEINIT)
8 GTF issues the SETEVENT macro instruction with ADD specified as the action so that the appropriate MC tables will be constructed for all EIDs and classes of events to be traced.	AHLGTFI AHLSETEV	(MCQEINIT)

GTF Diagram 2. Program Initialization (Part 7 of 8)



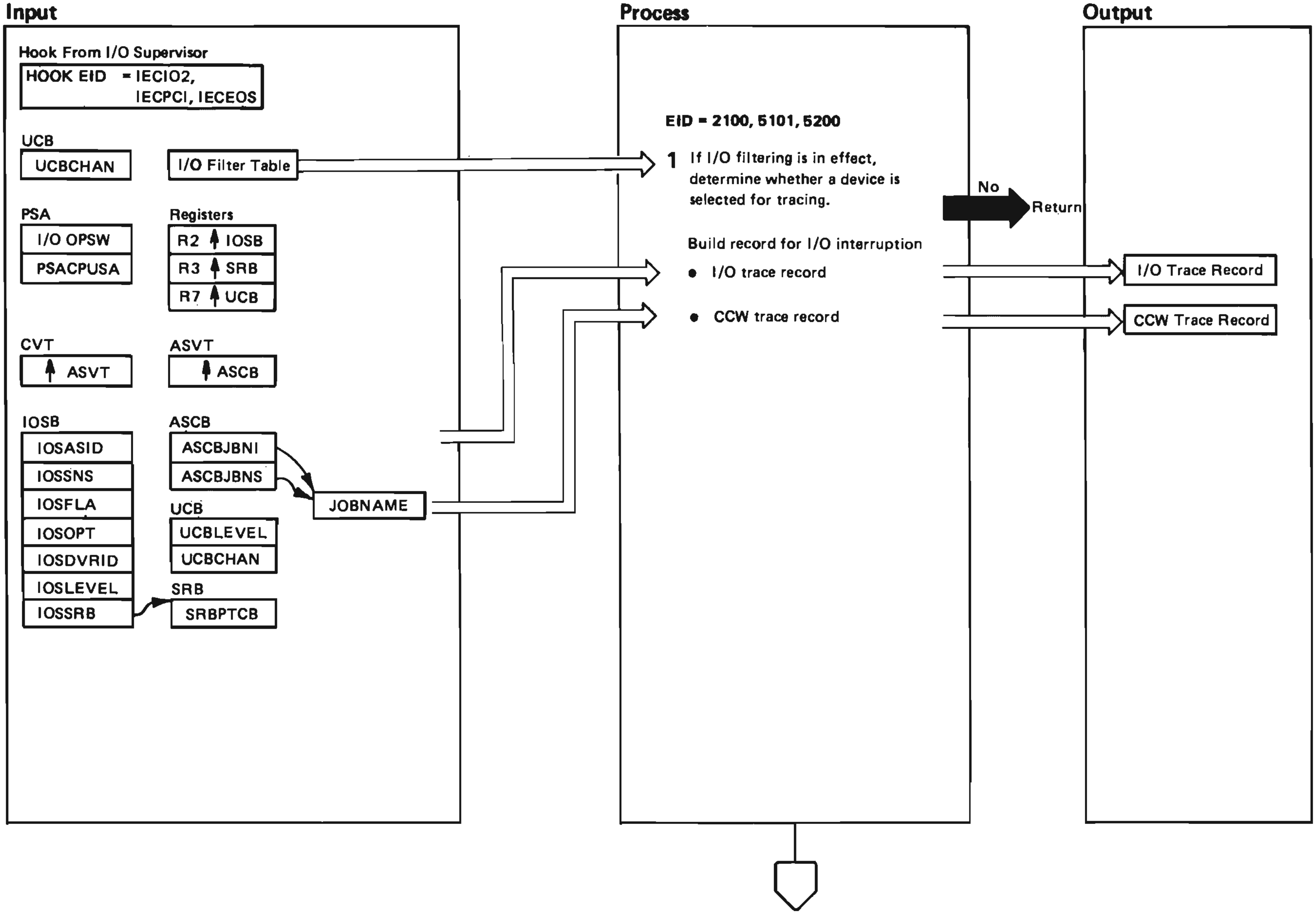
GTF Diagram 2. Program Initialization (Part 8 of 8)

Extended Description	Module	Label
9 GTF attaches the writer subtask to initialize the trace record blocking and buffering areas. (GTF determines which of three writers to attach based on whether trace data will be maintained internally or written to tape or to direct access.) The writer subtask issues GETMAIN macro instructions, first for an SRB to be used in buffering, then for the record-blocking areas (GFTBLOKs). The SRB has the addresses of the GTF ASCB and GTF's writer-subtask TCB. The GTFBLOKs are obtained one at a time in SQA. (If the size of real storage is 768K, GTF obtains only 3 GTFBLOKs.) The first block to be filled is initialized with a use count and with the CVTTZ value. A pointer to it is set in the CURRBLOK field of the GTFPCT. Remaining blocks are all chained off the first block. The buffers and buffer control blocks (GTFBCBs) are obtained from subpools 5 and 6 respectively. The buffers required for the GTF history queue are chained (via pointers in the associated BCBs) from the HQHEAD field in GTFPCT. Each GTFBCB contains a use count which indicates the number of queues the GTFBCB is on. This use count is never less than 1, since buffers not currently in use are on the empty queue.	AHLGTFI AHLCWRIT AHLIWRIT AHLWWRIT	
GTF also attaches and initializes the subtask which performs asynchronous operator communication during unrecoverable errors in trace processing.	AHLGTFI AHLWTASK	
10 GTF reissues the SETEVENT macro Instruction specifying ACTIVATE to enable tracing after successful completion of the above. The SETEVENT service obtains work/save areas (MCAWSA) to be used by the MC routing facility. If the CCW option was requested, the SETEVENT service obtains AHLCCWWK. It also initializea the class mask in control register 8 for each active CPU.	AHLTMON AHLSETEV	

Extended Description	Module	Label
11 While GTF tracing is active, GTF waits on a list of ECBs which cause GTF termination when posted. These are the STOP ECB, termination ECB, the writer/WTASK error ECB, and the two ATTACH ECBs for the writer and WTO subtasks. The CVT bits which indicate that GTF is active are set on. GTF then receives control from the program check interruption handler whenever a hook is issued for an event enabled for tracing. Asynchronously, the writer and WTO subtasks wait on the I/O ECB and WTO ECBs, respectively. When either of these are posted, the writer and/or the WTO subtask perform the requested function.	AHLTMON	

Note: During initialization, GTF frees storage for its temporary tables when they are no longer needed. These include INITDATA, EIDLIST, the device tables, and RANGETAB. If the STOP ECB is posted during initialization, the program terminates after freeing all resources.

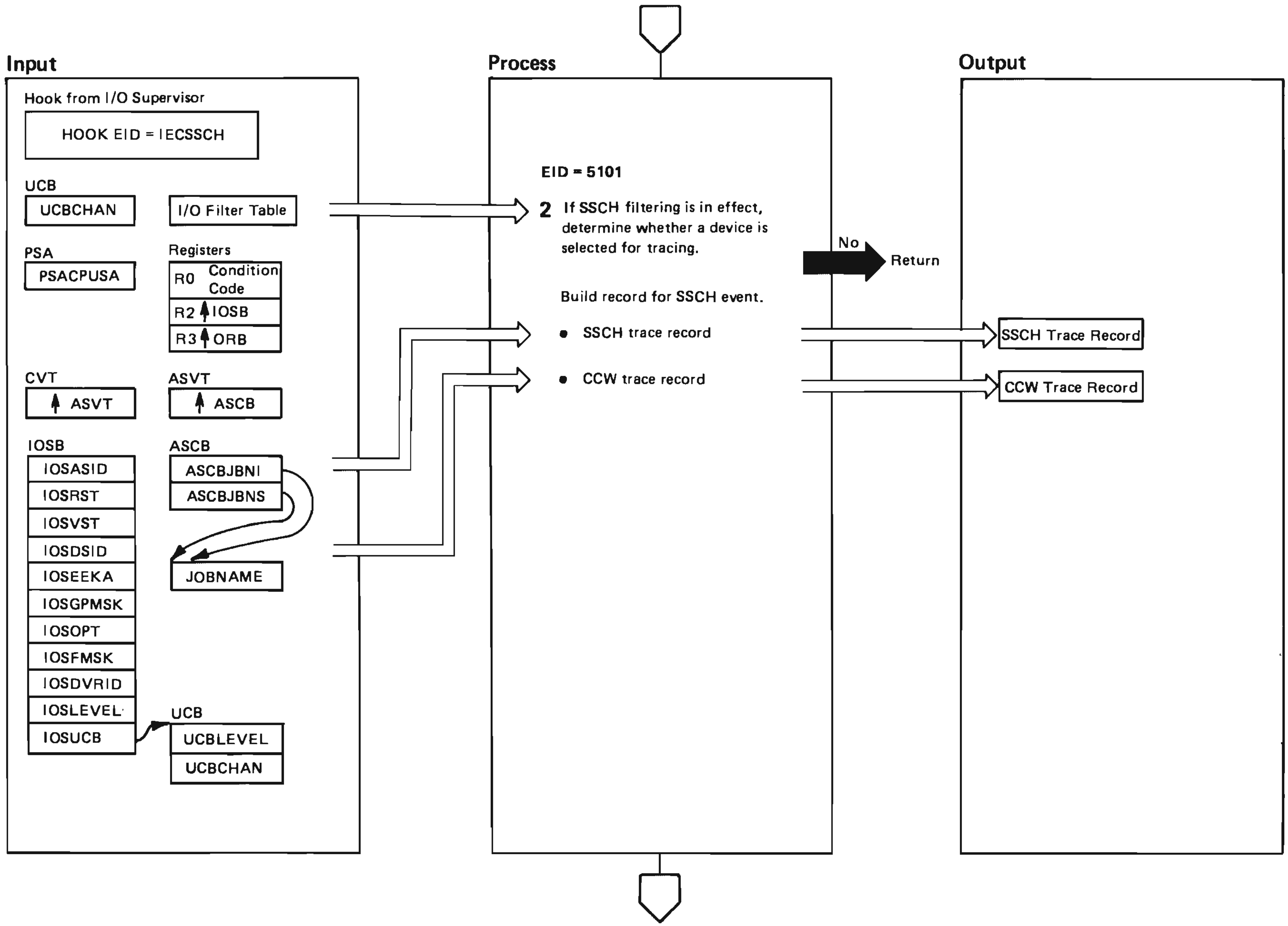
GTF Diagram 3. Hook Processing (Part 1 of 20)



GTF Diagram 3. Hook Processing (Part 2 of 20)

Extended Description	Module	Label
When the MC routing facility passes control to GTF for hook processing, GTF first determines if the event belongs to a specified set of jobs or address spaces. If so, then GTF performs event filtering (if selective tracing is in effect) or it gathers data to construct a record for the hook. The record is built in the GTF MC application work/save area (MCAWSA). The GTF issues an AHLSTACK macro instruction to cause the record prefix to be added and the record to be placed in a GTFBLOK. Full GTFBLOKs are copied into buffers. (See Diagram 4 for details.) In the case of a control hook from ABEND or from SVC Dump, the placement of the record in a block causes GTF to immediately copy the block to a buffer whether the block was full or not.	AHLTSELF AHLSBUF AHLSBLOK	AHLTSELF AHLSFEOB
1 The I/O supervisor issues a HOOK macro instruction for three types of I/O interruptions:		
• For end-of-sense interruption processing, it uses an EID of IECEOS.		
• For intermediate status interruption processing, it uses an EID of IECPCI.		
• For I/O interruptions with a valid UCB, it uses an EID of IECIO2.		
If interruptions for only certain devices are to be traced, GTF checks the I/O filter table to determine whether the current interruption is for one of the selected devices.		
• If TRACE = $\left\{\begin{matrix}\text{SYSM}\\\text{SYS}\\\text{SYSP}\\\text{IO}\\\text{IOP}\end{matrix}\right\}$, GTF builds an I/O trace record.	AHLTFCG	IOTRCE
• If TRACE = $\left\{\begin{matrix}\text{CCW}\\\text{CCWP}\end{matrix}\right\}$ and TRACE = $\left\{\begin{matrix}\text{IO}\\\text{IOP}\end{matrix}\right\}$, GTF builds CCW trace records in addition to the IO trace record.	AHLTCCWG	AHLTCCWG
Information for the records comes from various system data areas; the diagram shows these areas. The formats of the output records are in the 'Data Areas' section of this chapter.		

GTF Diagram 3. Hook Processing (Part 3 of 20)

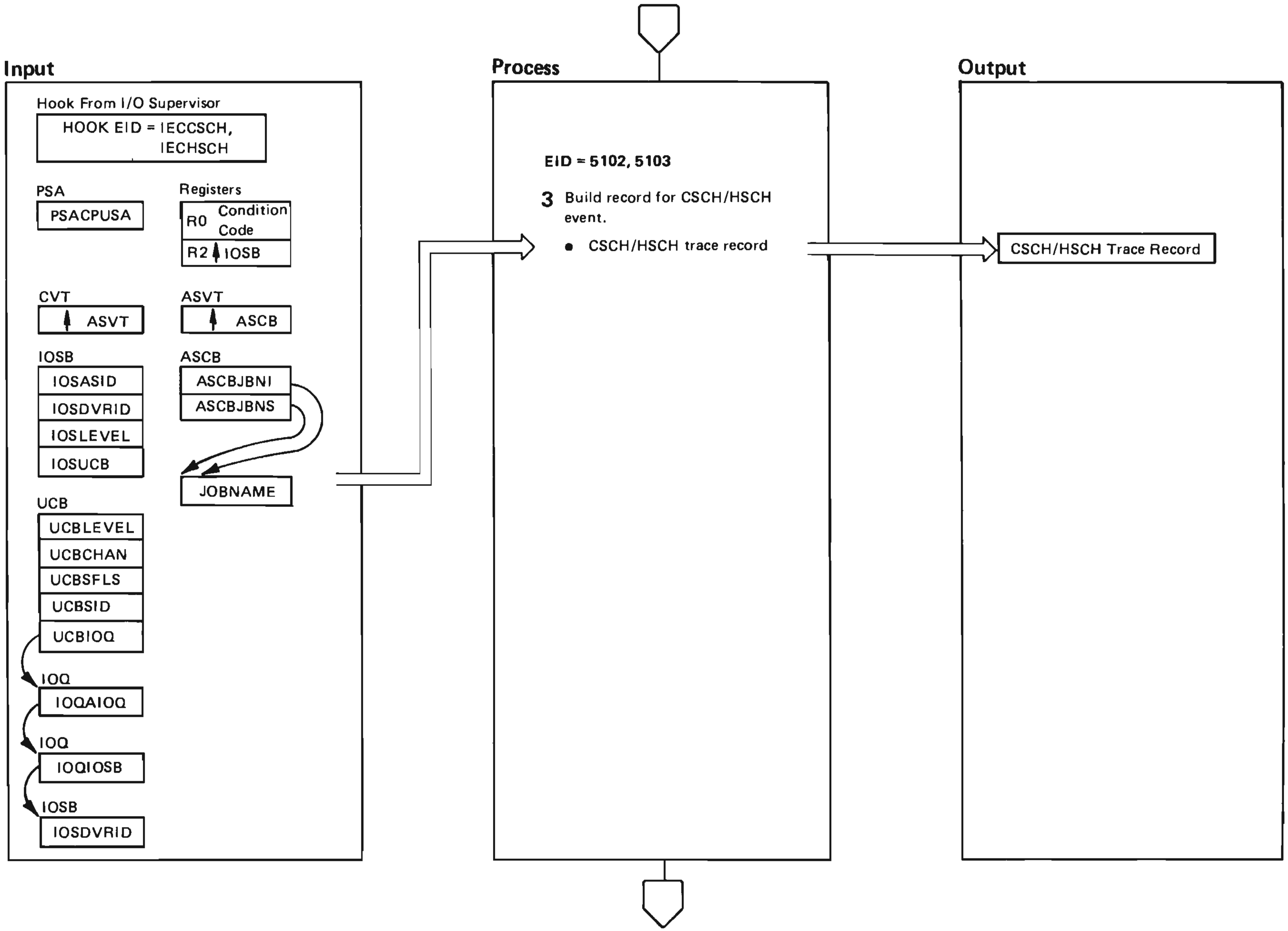


GTF Diagram 3. Hook Processing (Part 4 of 20)

Extended Description	Module	Label
2 The I/O supervisor issues a HOOK macro instruction with an EID of IECSSCH for all SSCH events in the IOS SSCH subroutine. If SSCH events for only certain devices are to be traced, GTF checks the SSCH filter table to determine whether the current event is for one of the selected devices.		
• If TRACE = {SYSM, SYS, SYSP, SSCH, SSCHP} GTF builds an SSCH trace record.	AHLTFCG	SSTRCE RSTRCE
• If TRACE = {CCW, CCWP} and TRACE = {SSCH, SSCHP}, GTF builds CCW trace records in addition to the SSCH trace record.	AHLTCCNG	AHLTCCWG

Information for the records comes from various system data areas; the diagram shows these areas. The formats of the output records are in the 'Data Areas' section of this chapter.

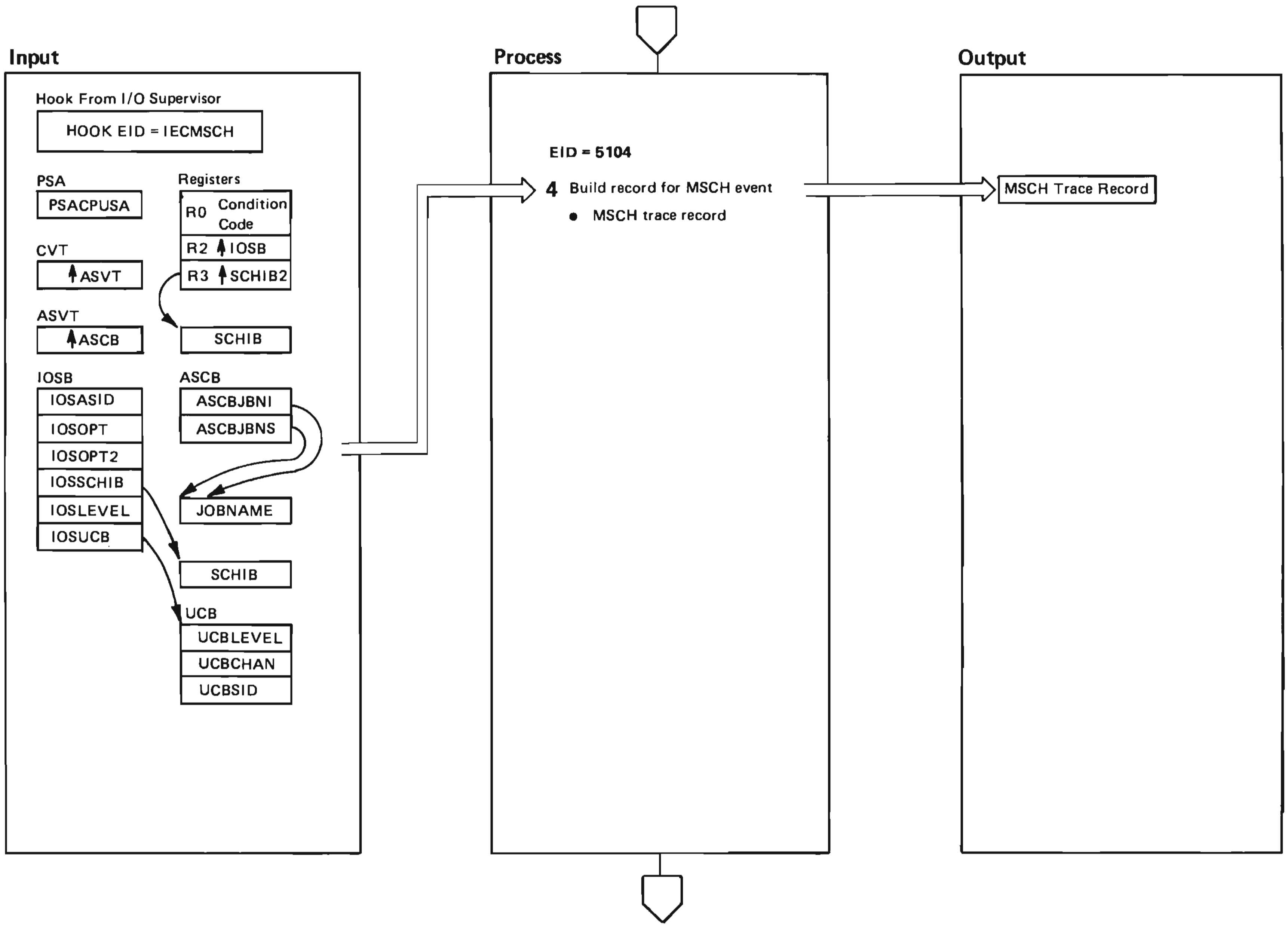
GTF Diagram 3. Hook Processing (Part 5 of 20)



GTF Diagram 3. Hook Processing (Part 6 of 20)

Extended Description	Module	Label
3 The I/O supervisor issues a HOOK macro instruction for clear subchannel and halt subchannel events:		
• For clear subchannel event processing, it issues an EID of IECCSCH.		
• For halt subchannel event processing, it issues an EID of IECHSCH.		
If TRACE = {SYSM, SYS, SYSP, CSCH, HSCH}, GTF builds a CSCH/HSCH trace record.	AHLTFCG	CHTRC
Information for these records comes from various system data areas; the diagram shows these areas. The formats of the output records are in the Data Areas section of this chapter.		

GTF Diagram 3. Hook Processing (Part 7 of 20)

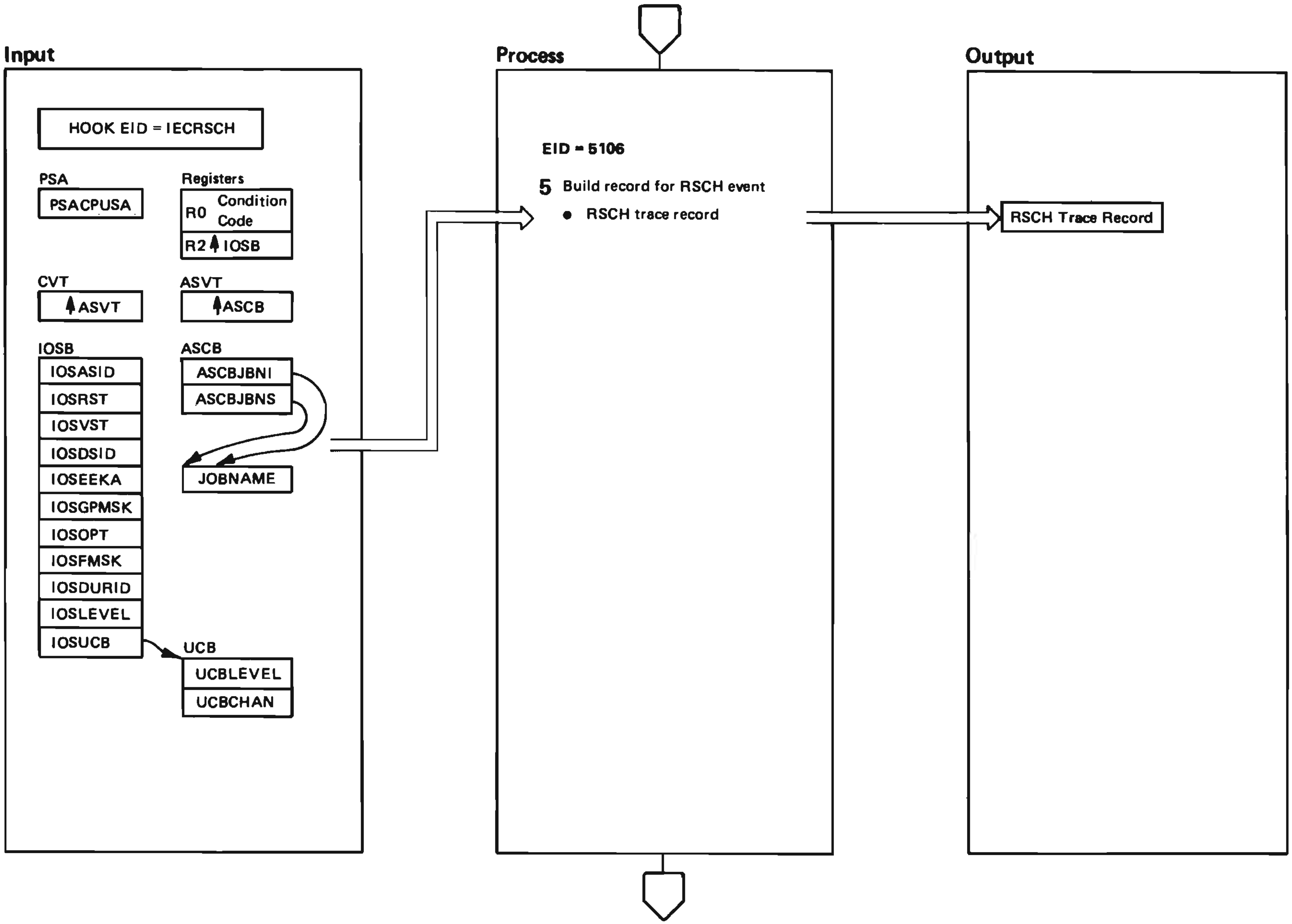


GTF Diagram 3. Hook Processing (Part 8 of 20)

Extended Description	Module	Label
4 The I/O supervisor issues a HOOK macro instruction with an EID of IECMSCH for all MSCH events:		
• If TRACE = $\left\{\begin{array}{l}\text{SYSM}\\\text{SYS}\\\text{SYSP}\\\text{MSCH}\end{array}\right\}$, GTF builds an MSCH trace record.	AHLTFCG	MSTRCE

Information for these records comes from various system data areas; the diagram shows these areas. The formats of the output records are in the Data Areas section of this chapter.

GTF Diagram 3. Hook Processing (Part 9 of 20)

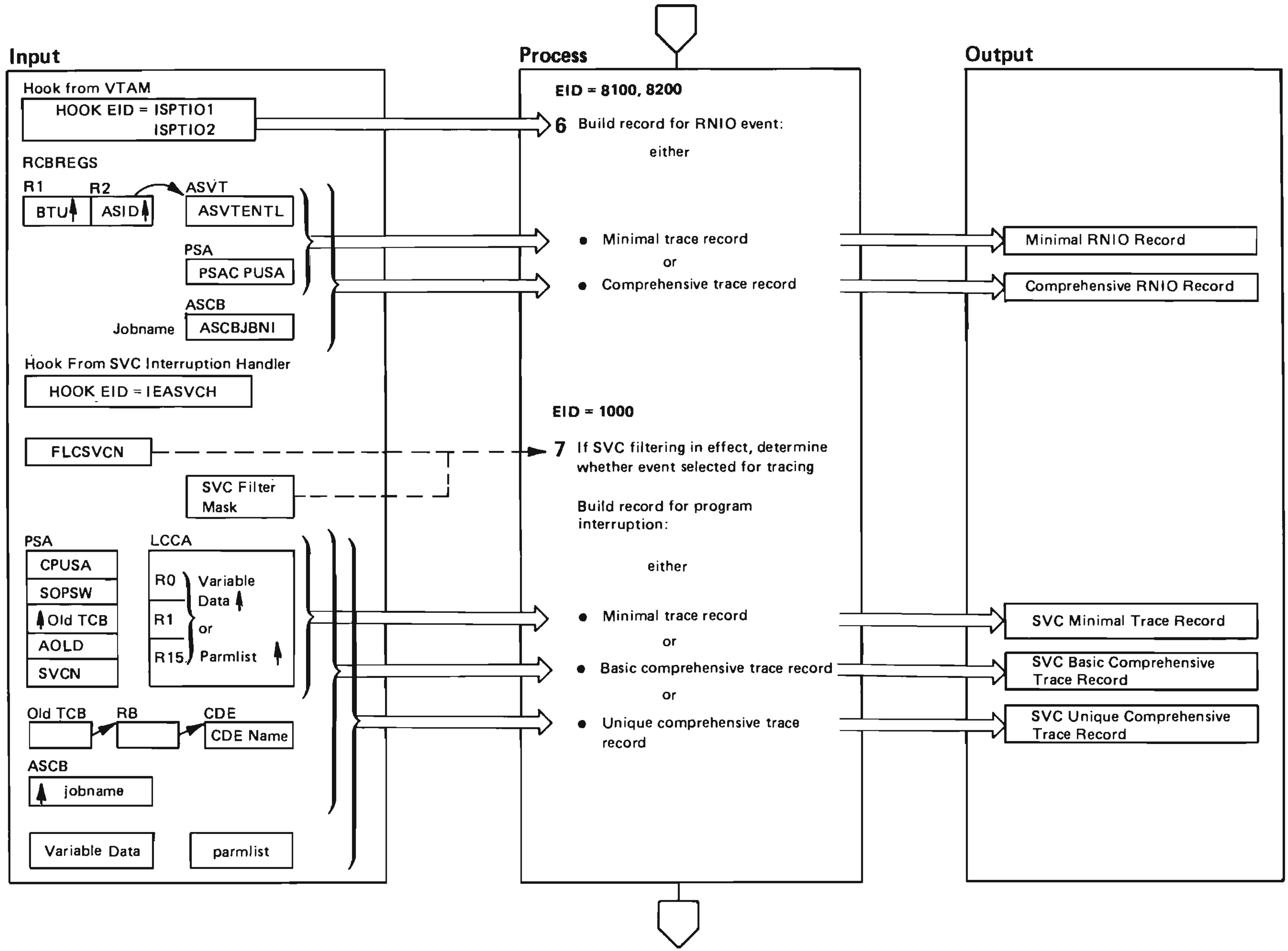


GTF Diagram 3. Hook Processing (Part 10 of 20)

Extended Description	Module	Label
5 The I/O supervisor issues a HOOK macro instruction with an EID of IECRSCH for all RSCH events:		
• If TRACE = {SYSM, SYS, SYSP, SSCH, SSCHP}, GTF builds an RSCH trace record.	AHLTFCG	RSTRC

Information for these records comes from various system data areas; the diagram shows these areas. The formats of the output records are in the Data Areas section of this chapter.

GTF Diagram 3. Hook Processing (Part 11 of 20)



GTF Diagram 3. Hook Processing (Part 12 of 20)

Extended Description	Module	Label
6 VTAM issues a HOOK macro instruction with an EID of ISPTPIO1 or ISPTPIO2 for every input or output BTU. GTF builds one of two records depending on the GTF options in effect:		
• If MODE=INT or TRACE=SYSM, RNIO, the program builds an RNIO minimal trace record.	AHLTVTAM	AHLACFV
• If TRACE=RNIO, GTF builds an RNIO comprehensive trace record.	AHLTVTAM	AHLACFV
Information for the records comes from various system data areas, shown in the diagram. The output record formats are described in the "Data Areas" section.		
7 The SVC interruption handler issues a HOOK macro instruction with an EID of IEASVCH for all SVC interruptions. If interruptions for only certain SVC numbers are to be traced, GTF checks the SVC filter table to determine whether the current interruption is for a selected number. GTF builds one of three records, depending on the GTF options in effect and the SVC number:	AHLTSYFL	AHLFSVC
• If TRACE=SYSM, GTF builds an SVC minimal trace record.	AHLTSYSM	AHLSVC
• If TRACE = $\left\{\begin{array}{l}\text{SYS}\\\text{SYSP}\\\text{SVC}\\\text{SVCP}\end{array}\right\}$, GTF builds a basic SVC comprehensive trace record.	AHLTSVC	AHLTSVC
• In addition to the basic comprehensive trace information, for certain SVC numbers GTF adds unique information to the record.	AHLTSVC	AHLTSVC
Information for the records comes from various system data areas, shown in the diagram. The output record formats are described in the "Data Areas" section.		

GTF Diagram 3. Hook Processing (Part 13 of 20)

Input

Hook From Dispatcher.

HOOK EID = IEADISP1,IEADISP2
IEADISP3,IEADISP4

PSA
- CPUSA
- ASCB
- PSW
- Old TCB

LCCA
- R0
- R1
- R15
- Global/Local Indicator

CVT
- ASVT

ASVT
- ASCB

ASCB
- Jobname

Old TCB

RB

CDE
- CDE Name

Hook from EXT PFLIH

HOOK EID = IEAEINT

PSA
- CPUID
- ASCB
- PCCA
- LCCA
- TCB
- parms
- Physical CPU
- Interruption Code

LCCA
- Old PSW
- R1
- R15

TCB
- TQE

or PCCA
- TQE

ASCB
- jobname

TQE
- TCB
- ASCB
- FLAGS
- TQEEXIT

Process

EID = 0001, 0002, 0003, 0004

8 Build record for TCB, LSR, SRB, or SVC prolog dispatching event

either

- Minimal trace record

or

- Comprehensive trace record

EID = 6201

9 Build record for external interruption

either

- Minimal trace record

or

- Comprehensive trace record

Output

DSP Minimal Trace Record

DSP Comprehensive Trace Record

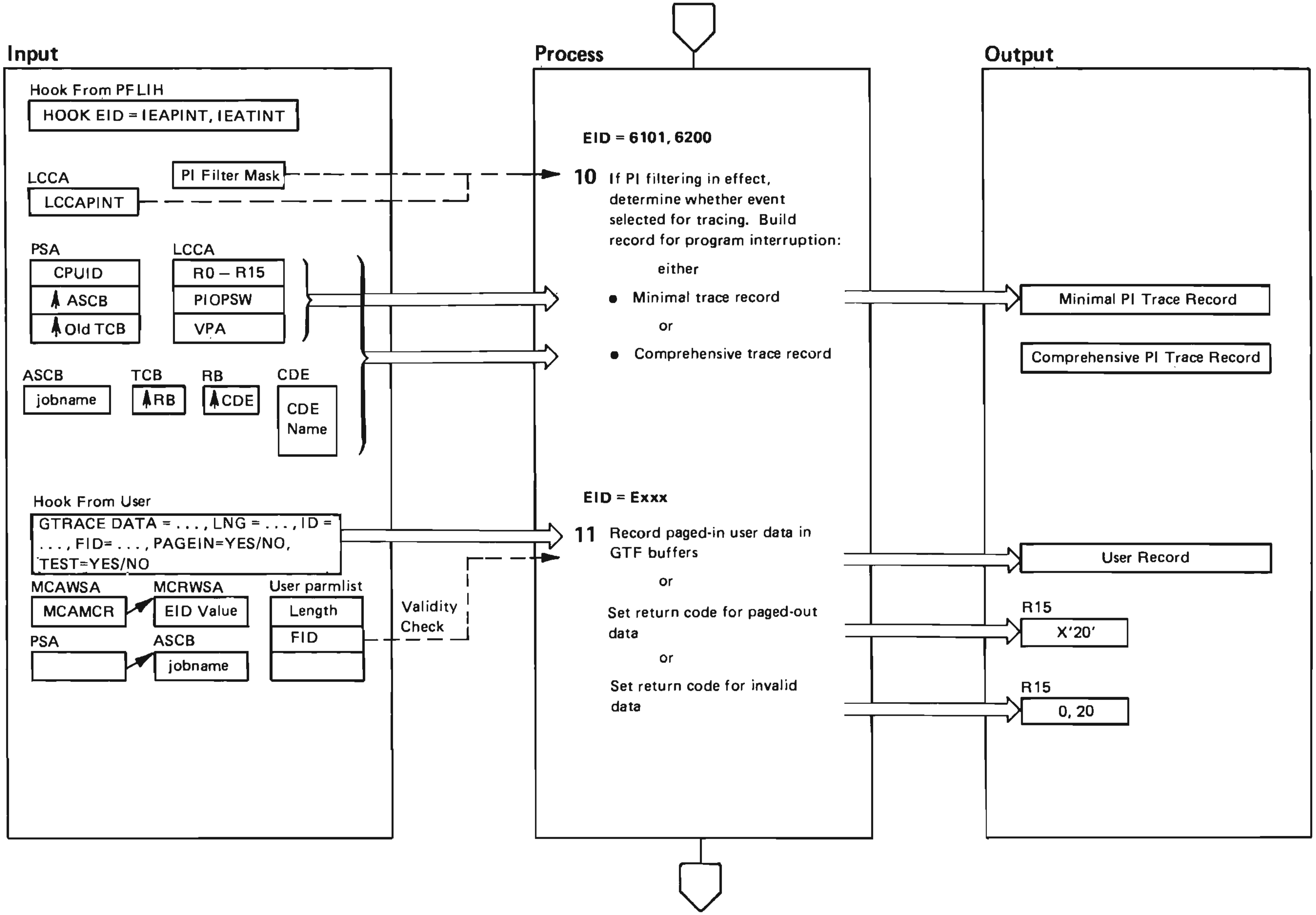
EXT Minimal Trace Record

EXT Comprehensive Trace Record

GTF Diagram 3. Hook Processing (Part 14 of 20)

Extended Description	Module	Label
8 The dispatcher issues a HOOK macro instruction when dispatching a unit of work:		
• For an SRB, it uses an EID of IEADISP1.		
• For an LSR, it uses an EID of IEADISP2.		
• For a TCB, it uses an EID of IEADISP3.		
The SVC exit prolog issues a HOOK macro instruction with an EID of IEADISP4 when dispatching a task. GTF builds one of two records:		
• If TRACE=SYSM, DSP, GTF builds a DSP minimal trace record.	AHLTXSYS	AHLDSP
• If TRACE=DSP, GTF builds a DSP comprehensive trace record.	AHLTPID	AHLTDSP AHLTSRB AHLTLSR
Information for the records comes from various system data areas shown in the diagram. The output record formats are described in the "Data Areas" section.		
9 The external FLIH issues a HOOK macro instruction with an EID of IEAEINT for all external interruptions. GTF builds one of two records:		
• If TRACE=SYSM, GTF builds an EXT minimal trace record.	AHLTSYSM	AHLEXT
• If TRACE=$\left\{\begin{array}{l}\text{SYS}\\ \text{SYSP}\\ \text{EXT}\end{array}\right\}$, GTF builds an EXT comprehensive trace record.	AHLTEXT	AHLTEXT
Information for the records comes from system data areas, shown in the diagram. The output record formats are described in the "Data Areas" section.		

GTF Diagram 3. Hook Processing (Part 15 of 20)



GTF Diagram 3. Hook Processing (Part 16 of 20)

Extended Description	Module	Label
10 The PFLIH issues a branch-type HOOK macro instruction with an EID of IEAPINT for all program interruptions with codes 1-17, 19. It uses an EID of IEATINT for PI 18. If interruptions for only certain interruption codes are to be traced, GTF checks the PI filter mask to determine whether the current interruption was selected for tracing. GTF builds one of two records:	AHLTSYFL	AHLFPI
• If TRACE=SYSM, GTF builds a PI minimal trace record.	AHLTSYSM	AHLPI
• If TRACE= $\left\{\begin{array}{l}\text{SYS}\\ \text{SYSP}\\ \text{PI}\\ \text{PIP}\end{array}\right\}$, GTF builds a PI comprehensive trace record.	AHLTPID	AHLTPI
Information for the records comes from system data areas shown in the diagram. The output record formats are described in the "Data Areas" section.		
11 The IBM user or component issues a GTRACE macro instruction to have GTF place a user/component record in the GTF buffers.		
• If TRACE=USR or TRACE=USRP is in effect, GTF accepts the record. GTF checks the validity of the data passed by the user/component and moves it to the GTF buffer.		
If the user specified the PAGEIN=YES keyword on the GTRACE macro instruction, code generated by expansion of the macro causes the trace data to be paged in. If the user specified the PAGEIN=NO keyword on the GTRACE macro instruction and a page fault occurs in the access of user data, GTF sets a return code of X'20' and no record is built.	AHLTUSR	AHLTUSRE

GTF Diagram 3. Hook Processing (Part 17 of 20)

Input

Hook From Systems Resource Manager
EID = IEASRM

PSA
- CPUID
- Old ASCB

LCCA
- R0 (ASID)
- R1
- R15
- PSW

CVT
- ASVT

ASVT
- ASCB

ASCB
- Jobname

Hook From Recovery/Termination Manager
EID = IEAFRR, IEASTAE

PSA
- CPUID
- ASCB

R1
- RTCA

RTCA
- Error PSW
- Completion Code
- Flags
- Return Code
- Retry Address

Hook From ABEND
- HOOK EID = IEAABOF

PSA
- PSAAOLD

LCCA
- R1 (TCB)

ASCB
- ASID

Hook From SVC Dump
- HOOK EID = IEASVCD

PSA
- PSAAOLD

ASCB
- ASID

Process

EID = 4001

12 Build record for SRM invocation:

either
- Minimal trace record

or
- Comprehensive trace record

EID = 4002, 4003

13 Build record for RR event:

either
- Minimal trace record

or
- Comprehensive trace record

EID = F200

14 Build ABDUMP record.

EID = F100

15 Build SVC dump record.

Output

- Minimal SRM Trace Record
- Comprehensive SRM Trace Record
- Minimal RR Trace Record
- Comprehensive RR Trace Record
- Save Hook Control Record
- Save Hook Control Record

Extended Description	Module	Label
12 The system resources manager issues a HOOK macro instruction each time it is entered. GTF builds one of two records:		
• If TRACE=SYSM, SRM, then GTF builds an SRM minimal trace record.	AHLTXSYS	AHLSRM
• If TRACE=SRM, then GTF builds an SRM comprehensive trace record.	AHLTFOR	AHLTSRM
13 The recovery/termination manager issues a HOOK macro instruction on return from each FRR, STAE, and ESTAE recovery routine. GTF builds one of two different records, depending on the options in effect:		
• If TRACE=SYSM, then GTF builds an RR minimal trace record.	AHLTSYSM	AHLFRR
• If TRACE=RR, then GTF builds an RR comprehensive trace record.	AHLTFOR	AHLTFRR
14 The ABEND/SNAP routine issues a HOOK macro instruction if SDATA=TRT was specified. GTF builds a control record specifying that buffer contents are to be saved for dumping. See Diagrams 4 and 6.	AHLTDIR	
15 The SVC Dump routine issues a HOOK macro instruction if SDATA=TRT was specified. GTF builds a control record specifying that buffer contents are to be saved for dumping. See Diagrams 4 and 6.	AHLTDIR	

GTF Diagram 3. Hook Processing (Part 19 of 20)

Input

Hook From SLIP Processor

EID-IEAVSLSD, IEAVSLSU, IEAVSLUR

MCRWSA
- ↑ PLIST
- EID

PSA
- CPUID
- ↑ ASCB
- ↑ TCB
- ↑ LCCA

TCB
- ↑ JSCB
- ↑ CDE

JSCB
- Program Name

CVT
- ↑ CDE

CDE
- Module Name
- ↑ XTLST

LCCA
- Old PSW
- PIC and ILC
- PER Interrupt Code

PLIST
- ↑ SCE
- ↑ Registers
- ↑ TRDATA Information (in SCVA)
- MODE/COND Flags
- ↑ DEBUG Data
- ↑ IEAVTADR
- Data Unavailable Counter

ASCB
- ASID
- ↑ jobname

SCE
- TRAP ID
- ↑ SCVA
- PER Trap Mode

SCVA
- TRDATA Information (User-Defined)

CSCB
- JOBNAME

XTLST
- LEN
- ADDR

Process

EID = 4004, 4005, 4006

16 Build record for SLIP event

either

- SLIP standard trace record (EID=4004)

or

- Combined SLIP standard trace record and user or debug data (EID=4005)

or

- SLIP trace record with user-defined data only (EID=4006)

Return Next Instruction After MC

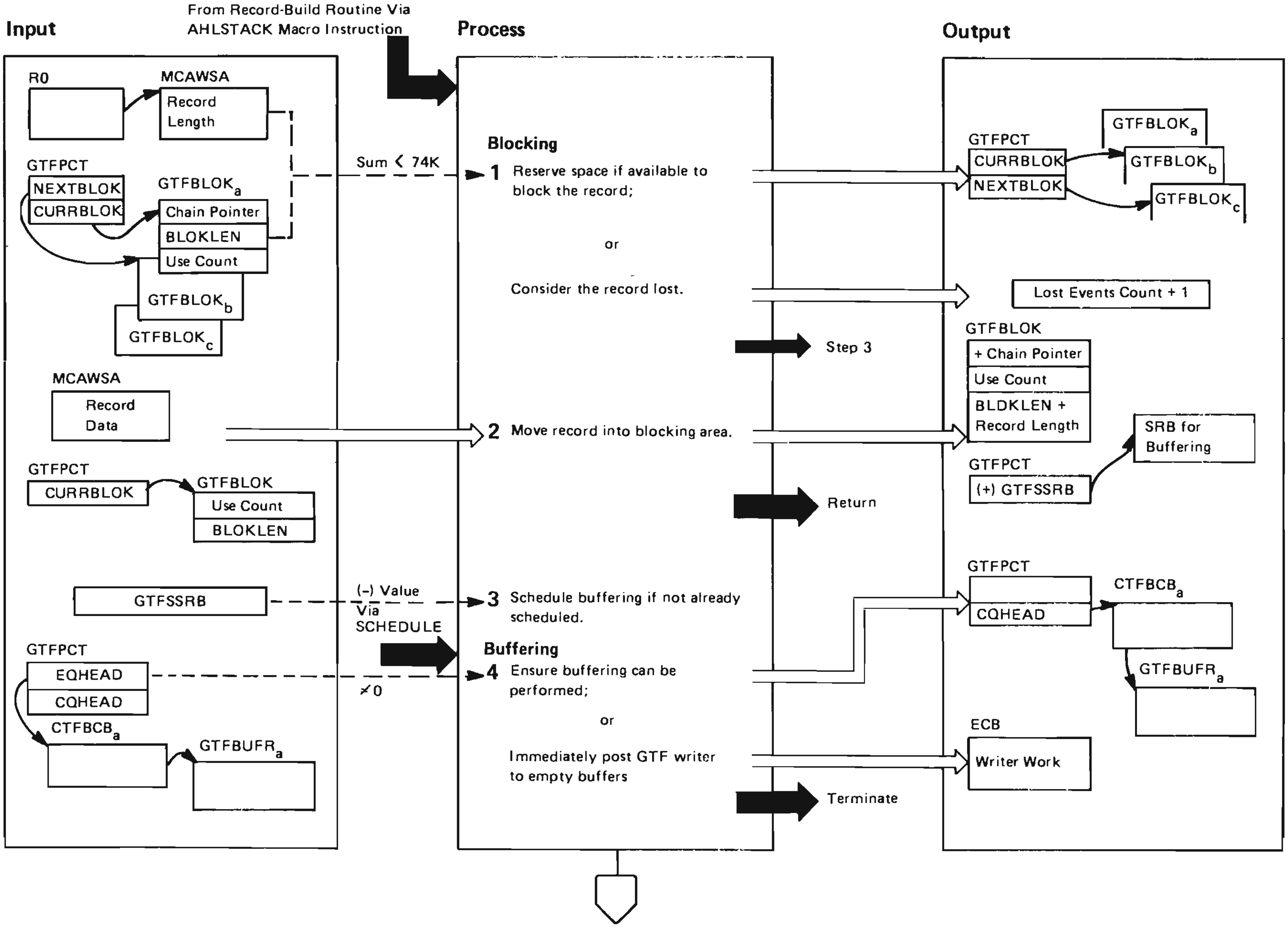
Output

- SLIP Standard Record
- SLIP Standard/User Record
  or
  SLIP DEBUG Record
- SLIP User Record

GTF Diagram 3. Hook Processing (Part 20 of 20)

Extended Description	Module	Label
16 The SLIP processor (IEAVTSLP) issues a HOOK macro instruction for:		
• A SLIP event that has been successfully matched with an enabled SLIP trap and either TRACE or TRDUMP is the specified action for the trap.		
• A SLIP trap that is in DEBUG mode and is inspected by the SLIP processor as a result of any SLIP event.		
GTF builds one of these records:		
• If the EID=X'4004', GTF builds a record containing standard information obtained by GTF from the data areas shown in the diagram.	AHLTSLIP	AHLSLSTD
• If the EID=X'4005', GTF builds a record containing SLIP standard information followed by the information that is (1) requested by the user via the TRDATA parameter of the SLIP command, or (2) supplied by IEAVTSLP when a SLIP trap is DEBUG mode.	AHLTSLIP	AHLSLSTD AHLSLUSR
• If the EID=X'4006', GTF builds a record containing information requested by the user via the TRDATA parameter of the SLIP command.	AHLTSLIP	AHLSLUSR
If the user-defined data requested is greater than 136 bytes (including length fields) for a standard/user record, or greater than 256 bytes (including length fields) for a user record, the record is truncated.		
The output record formats are described in the "Data Areas" section.		

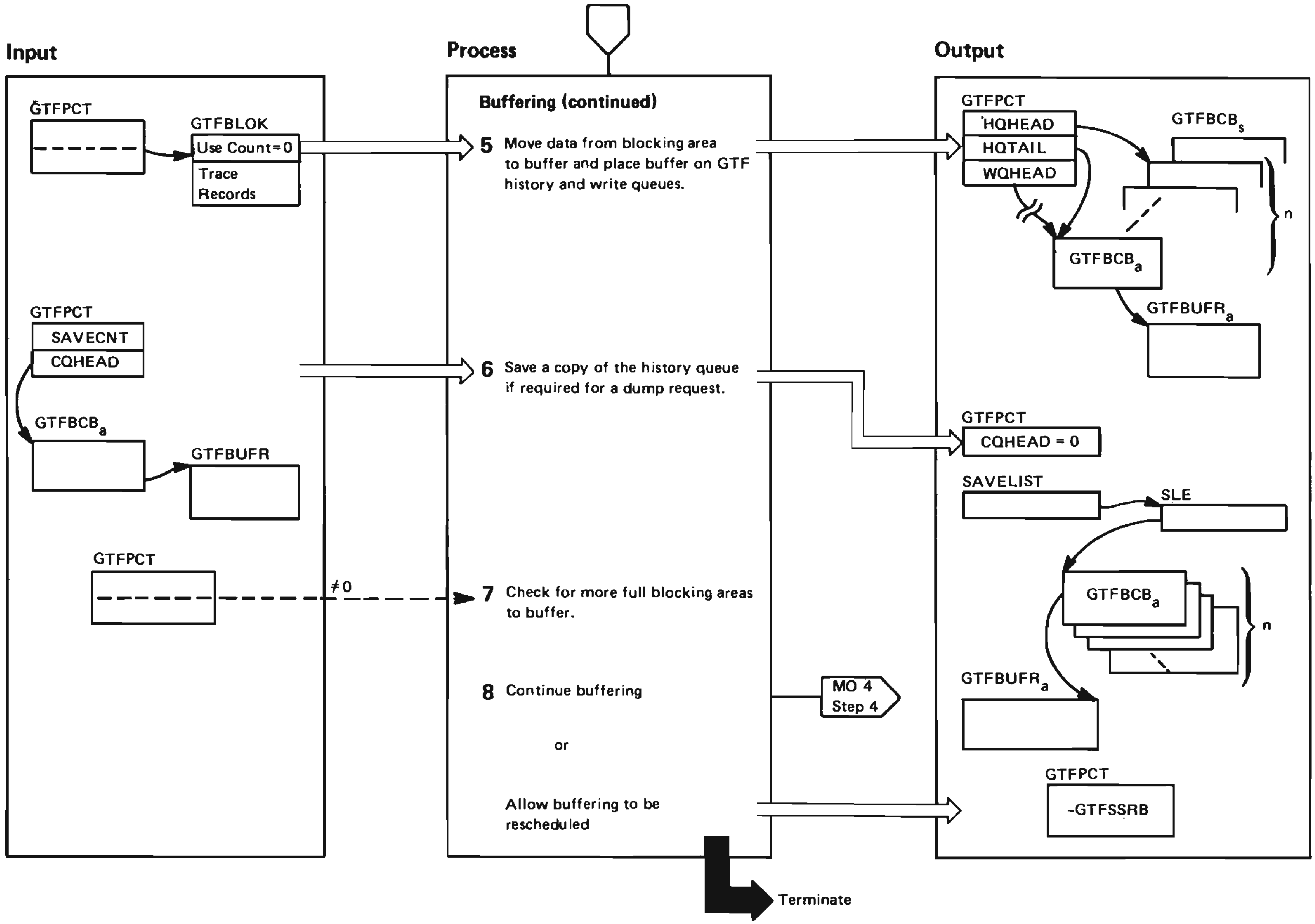
GTF Diagram 4. Trace-Data Blocking and Buffering (Part 1 of 4)



Extended Description	Module	Label
GTF manages the flow of trace data using both record-blocking areas (GTFBLOKs) and buffers (GTFBUFRs). Buffers are accessed through their buffer control blocks (GTFBCBs).		
Since multiprocessing will cause concurrent construction, blocking, and buffering of trace records, GTF uses a series of use counts and locks to serialize the use of these areas.		
GTF also maintains queues of buffers for different purposes. These are: • History queue — The n most recent GTF buffers where n is specified by the BUF=parameter to control the amount of trace data in a dump. See *Service Aids* for information on the specification of n. • Writer queue — Full buffers to be written to an external data set or released. • Release queue — Buffers for which a PGRLSE must be issued before they are placed on the empty queue. • Empty queue — Buffers available to receive blocked trace records. • Current queue — Buffer currently being filled. • SLE queue — Buffers being held for a dump routine.		
1 GTF checks the length of the constructed trace record. (The length is in the MC application work/save area (MCAWSA).) If there is insufficient space for the constructed record in the current GTFBLOK, the next empty GTFBLOK on the chain is made the current blocking area. The CURRBLOK pointer in the GTFPCT is updated.	AHLSBLOK (AHLSFEOB)	
If all the GTFBLOKs are full, GTF updates the lost-events count since the trace data in the record will not be saved. Then, GTF schedules the buffering function as in Step 3.	(AHLSFEOB)	

Extended Description	Module	Label
2 Before moving the record into the block, GTF increments the use count in the block to prevent an attempt to buffer the data during the blocking process. GTF reserves space in the block for the record by adding the length of the new record to the length of already-blocked data (BLOKLEN). This allows other records to be entered into the block concurrently. (For concurrent entry of records, the BLOKUSE count is incremented accordingly. Buffering only occurs when the count is 0.) GTF moves the record from the construction area in MCAWSA to the GTFBLOK. Then it decrements the use count.	AHLSBLOK	
3 GTF determines whether buffering is already scheduled by checking the GTFSSRB value in GTFPCT. If this value is negative, buffering is not currently scheduled so GTF issues the SCHEDULE macro instruction.	AHLSBLOK	
4 GTF checks the pointer to the empty buffer queue (EQHEAD) in the GTFPCT. If there are no empty buffers, GTF posts the writer's work ECB so that buffers can be made available. (See diagram 5 for writer processing.) Buffering terminates until reenabled by the writer.	AHLSBUF	

GTF Diagram 4. Trace-Data Blocking and Buffering (Part 3 of 4)



GTF Diagram 4. Trace-Data Blocking and Buffering (Part 4 of 4)

Extended Description	Module	Label
5 If buffers are available on the empty queue and there is a full GTFBLOK, GTF makes the first buffer the current one (CQHEAD). It copies the GTFBLOK into the buffer and places it on the end of the history queue (QTAIL). It then decrements the use count in the GTFBLOK. For every buffer placed on the end of the history queue, one must be removed from the head. The HQHEAD pointer indicates the buffer to be removed. The buffer removed is placed on the release queue for later page release by the GTF writer. From the release queue, the buffers are replaced on the empty queue.		
6 GTF scans the buffer for a record indicating a request for trace data for ABEND/SNAP or SVC Dump. For every such record, GTF builds a save list element (SLE) and decrements the save count. All the buffers currently on the history queue are also placed on the SLE to save them for the dump. After this, GTF removes the address of the buffer in the current queue pointer (CQHEAD).	AHLSBUF	
7 For any other full GTFBLOKs, the buffering process is repeated.	AHLSBUF	
8 GTF also posts the writer subtask to indicate a buffer has been added to the write queue.	AHLSBUF	

Diagram 5. GTF-Writer Processing (Part 1 of 2)

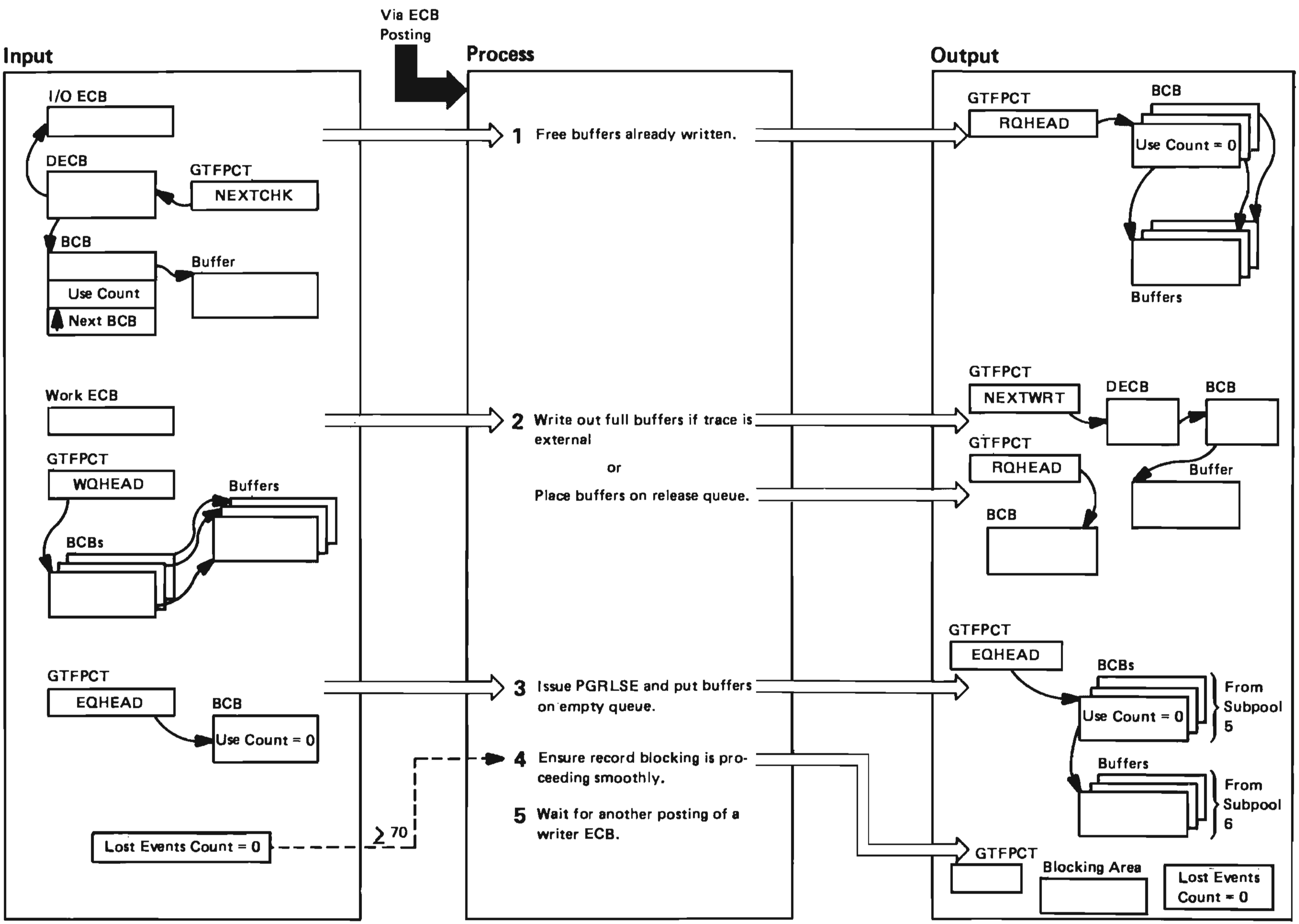
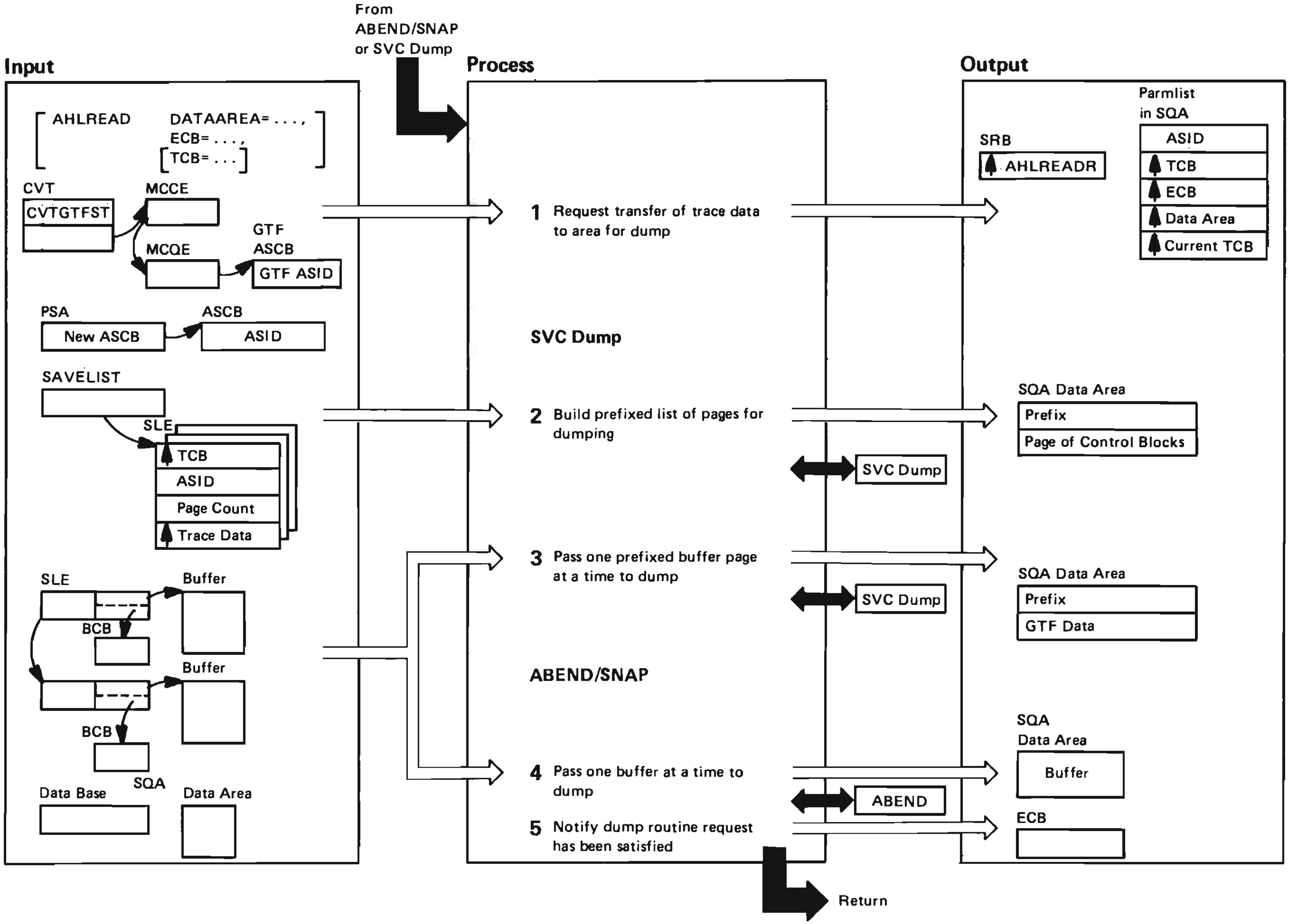


Diagram 5. GTF-Writer Processing (Part 2 of 2)

Extended Description	Module	Label
1 For external trace mode, the GTF writer subtask checks the ECB in the DECB pointed to by NEXTCHK. If it has been posted, a CHECK macro instruction is issued, and then the writer decrements the usecount field in the associated buffer control block (GTFBCB). If the use count is then zero, the writer puts the GTFBCB on the release queue and increments the count. GTF will subsequently issue a PGSER FREE macro instruction for the buffer page. The NEXTCHK pointer is updated to the chain address in the DECB. When the address in NEXTCHK is the same as the address in NEXTWRT, there are no more DECBs to check.	AHLCWRIT AHLWWRIT	
2 If the ECB was posted because buffers are full, processing depends on whether an external data set is defined (external trace mode). If there is not external data set, the writer removes the GTFBCB from the write queue. If there is an external data set, the writer removes a full buffer from the write queue and places it on an available DECB and issues a WRITE macro instruction with the buffer address as the write record.	AHLIWRIT AHLCWRIT AHLWWRIT	
For direct-access output, the writer checks the ECBs for a code of X'40' in the first byte. This code indicates that the end of the data set has been reached and wrapping is required. In this case, the next write is to the beginning of the data set.	AHLWWRIT	
3 The writer moves all buffers from the release queue to the empty queue. It issues a PGRLSE macro instruction for each buffer. It then checks the empty queue (EQHEAD) for available buffers. If there are insufficient buffers on the queue, the writer issues the GETMAIN macro instruction to obtain 4K buffer storage from subpool 6 and equivalent BCB storage from subpool 5. If the buffering routine has been disabled because of the lack of available buffers, the writer reenables it.	AHLCWRIT AHLIWRIT AHLWWRIT	
4 The writer checks the lost record count. If the count is 400 or greater, a blocking area (GTFBLOK) may need to be replaced.	AHLCWRIT AHLIWRIT AHLWWRIT	
5 After its processing, the writer subtasks reenters the wait state. Should the next ECB posted be the writer termination ECB, all queue activity is quiesced before termination. See MO 7 for details.	AHLIWRIT AHLCWRIT AHLWWRIT	

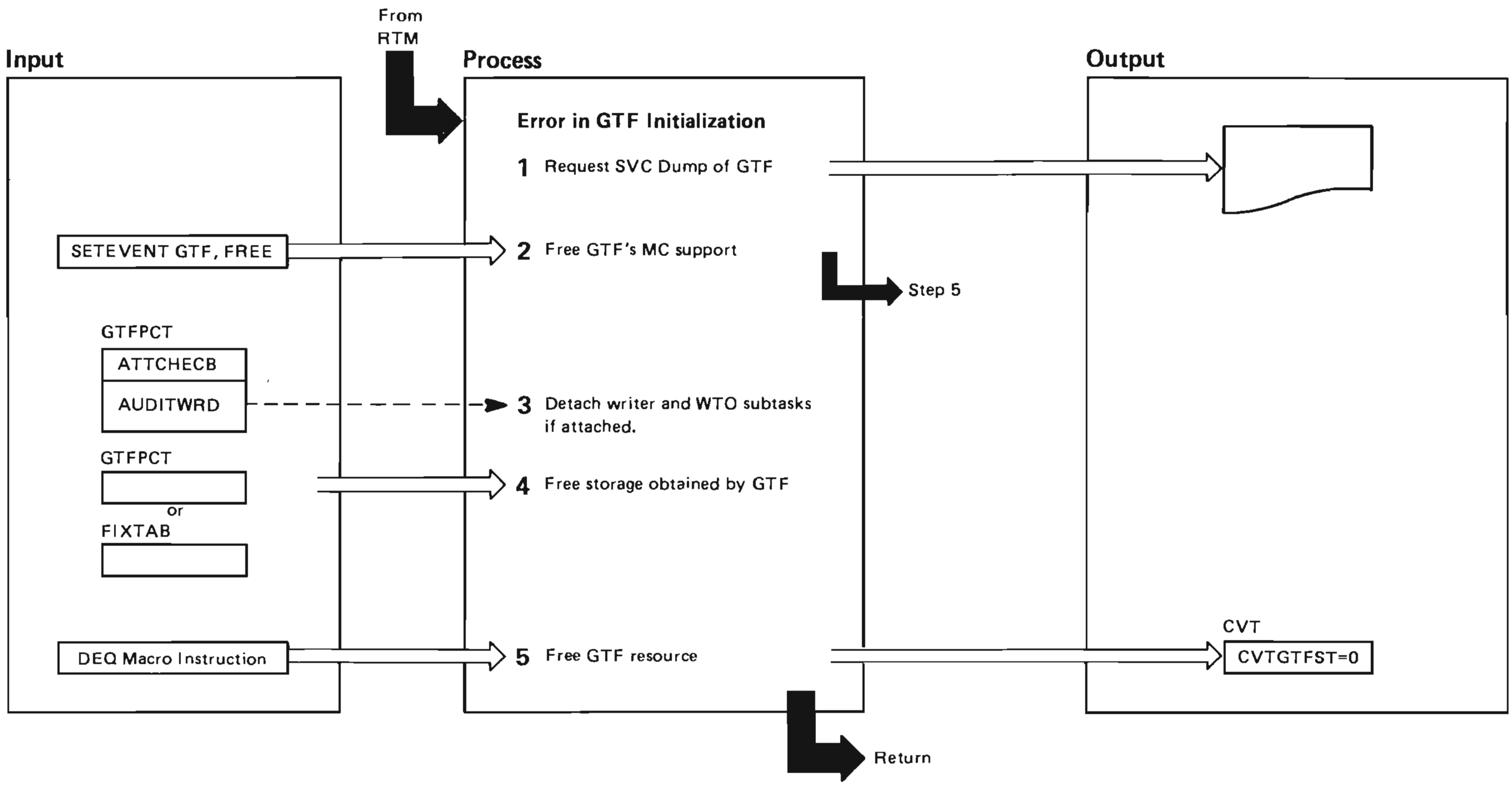
GTF Diagram 6. GTF/Dump Interface (Part 1 of 2)



GTF Diagram 6. GTF/Dump Interface (Part 2 of 2)

Extended Description	Module	Label
To provide trace data for ABEND/SNAP or SVC Dump, GTF saves the required data on a queue (SLE) until it can be dumped. This function is requested by HOOKs with EIDs of IEAABOF and IEASVCD for ABEND/SNAP and SVC Dump, respectively.		
1 When ABEND/SNAP or SVC Dump requires the trace data for a dump, it issues the AHLREAD macro instruction and then waits on an ECB. Expansion of AHLREAD causes a parameter list to be constructed in the data area provided by ABEND/SNAP or SVC Dump. An SRB to schedule the trace data transfer is constructed in subpool 245. Then the SRB is scheduled. However, if the address space being dumped is the GTF address space, no trace data will be passed to the dump program. This is indicated by a POST code for no data available.	AHLREAD Macro instruction	
2 For an SVC Dump request, GTF passes pages containing control blocks required by AMDPRDMP's edit function to format trace data. The page is prefixed with eight bytes of data required by AMDPRDMP. GTF passes one page of data for each AHLREAD macro instruction issued.	AHLREADR	
3 When SVC Dump requests subsequent pages of trace data, GTF builds the 8-byte prefix and passes the contents of a 4K buffer. If the address contained in the SLE is for a GTFBCB, GTF gets the buffer address from the GTFBCB.	AHLREADR	
4 GTF copies the buffer contents one at a time from the GTF address space to an area in SQA specified by ABEND/SNAP.	AHLREADR	
5 The ECB for the asynchronous data transfer is posted. The SAVELIST elements are then released. The records are then formatted by ABEND/SNAP (module IGC0F05A). User records are formatted if the user/component has provided a formatting routine for use by the dump service or the edit function of AMDPRDMP. If the user formatting routine cannot be loaded, ABEND/SNAP dumps these records in hexadecimal.	AHLxWRIT	

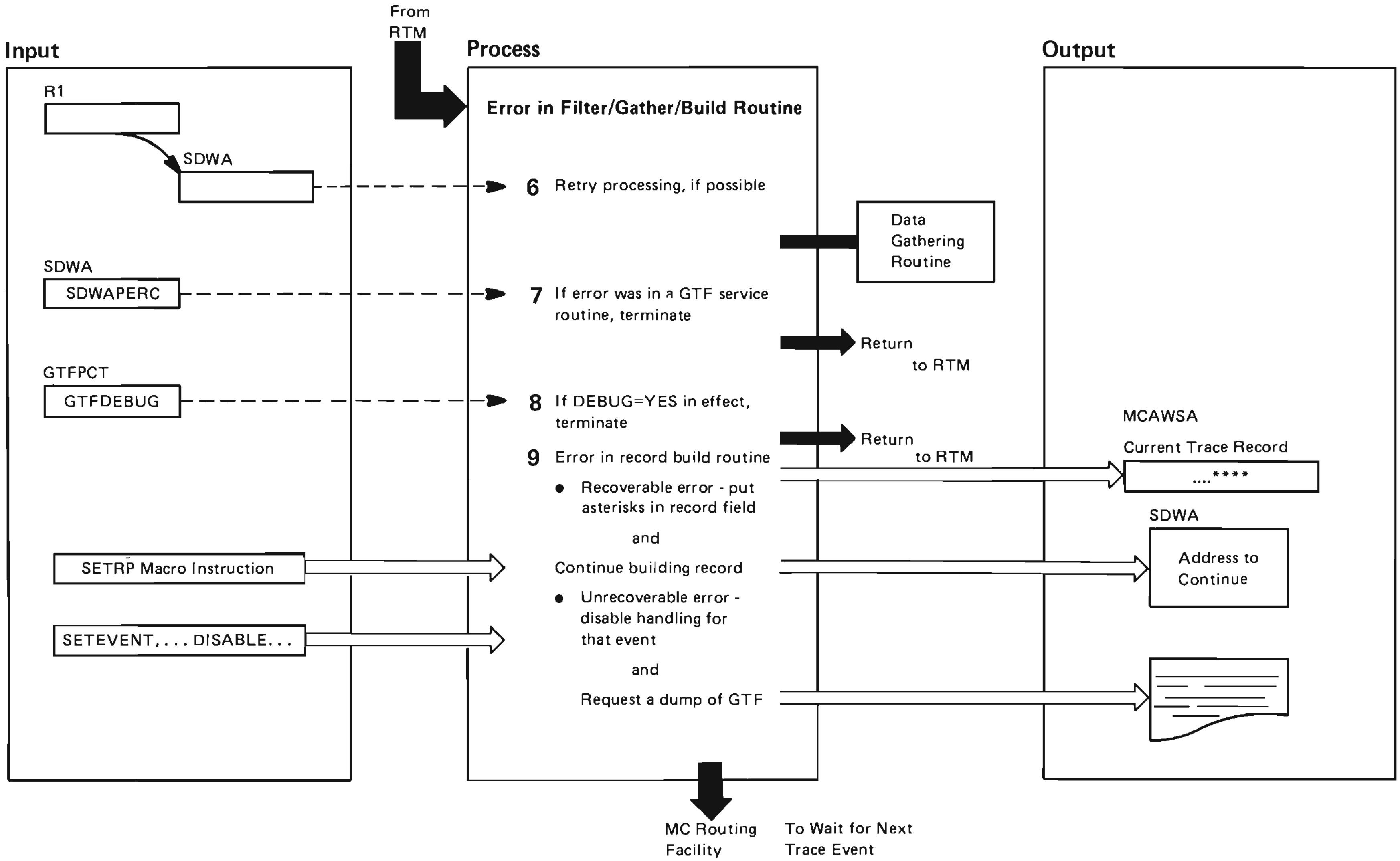
GTF Diagram 7. Error Recovery (Part 1 of 4)



GTF Diagram 7. Error Recovery (Part 2 of 4)

Extended Description	Module	Label
Error in GTF initialization		
1 When an error occurs during GTF initialization, the recovery/termination manager (RTM) passes control to the GTF initialization ESTAE routine. The ESTAE routine requests an SVC Dump of GTF and issues message AHL016I to inform the operator of the error.	AHLGTFI	AHLIESTA
2 Then a SETEVENT macro instruction specifying the FREE action is issued to free MC routing-facility support. See Section 6 for a description of the SETEVENT macro instruction.	AHLGTFI	AHLIESTA
3 After freeing MC routing-facility support, GTF posts the writer subtask for termination. It waits for the writer to complete its termination and then detaches the writer. The WTO subtask is detached in the same manner. If the GTFPCT tracking field (AUDITWRD) indicates that either subtask has not been attached yet, no processing for that subtask is performed.	AHLTMON or AHLGTFI	AHLTESTA or AHLIESTA
4 GTF frees any SQA storage it has obtained. It also issues the PGSER macro instruction to free all fixed storage in LPA occupied by the filter/gather/build routines. This storage is pointed to by FIXTAB.	AHLTMON or AHLGTFI	AHLTESTA or AHLIESTA
5 The GTF resource is then freed by issuing the DEQ macro instruction. GTF returns control to the system with a non-zero return code.	AHLTMON or AHLGTFI	AHLTESTA or AHLIESTA

GTF Diagram 7. Error Recovery (Part 3 of 4)



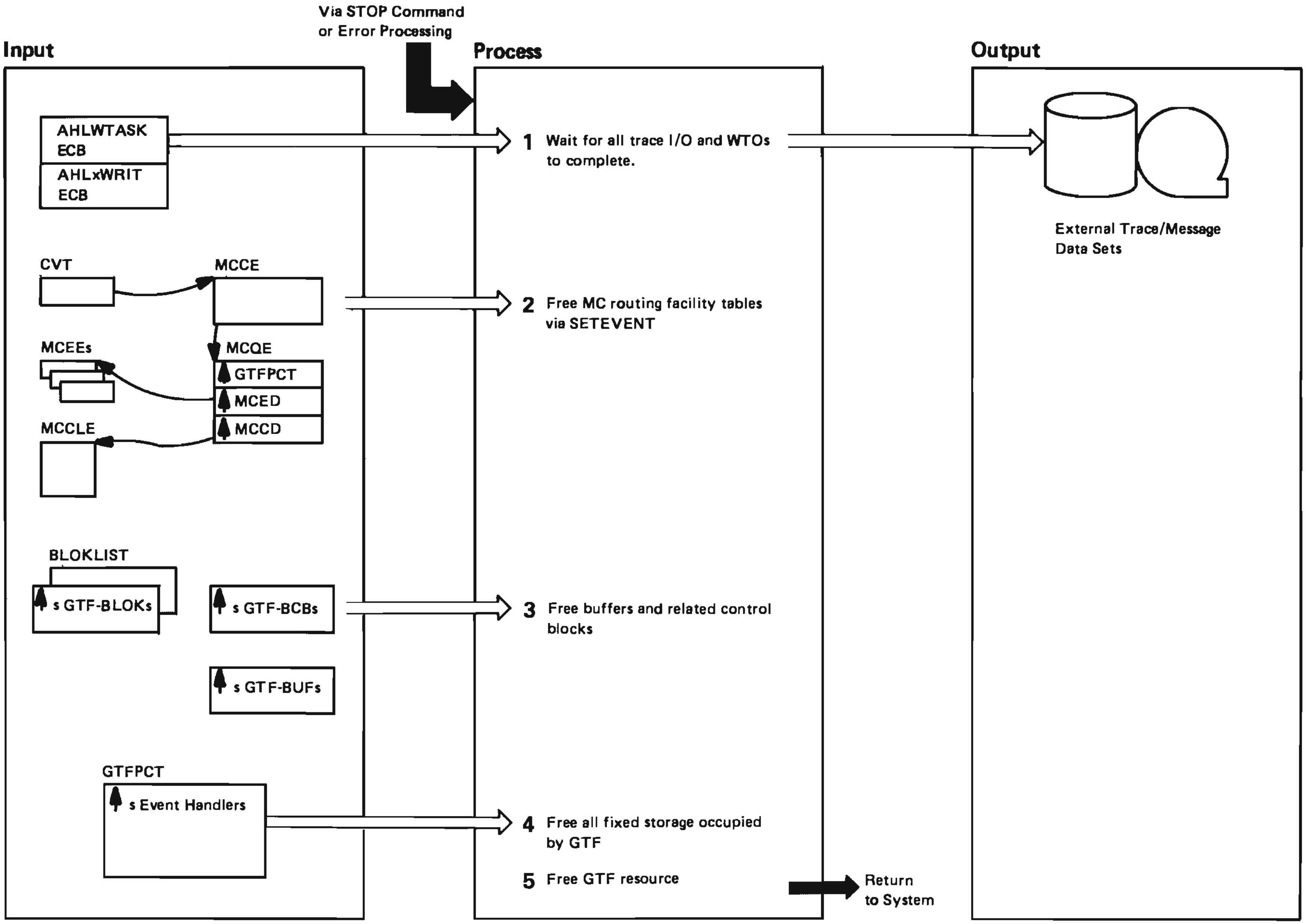
GTF Diagram 7. Error Recovery (Part 4 of 4)

Extended Description	Module	Label
Error in filter/gather/build routine		
6 If alternate CPU recovery (ACR) allows, GTF retries the error-causing processing. This is indicated by the STAE diagnostic work area (SDWA) passed to GTF by RTM.	Event handler's error routine	
7 If the error occurred in a service, for example, record blocking (AHLSTACK routine), GTF passes control to the recovery/termination manager for termination of GTF.		
8 If DEBUG=YES was specified in the START command, GTF passes control to the recovery/termination manager for enabled termination of GTF.		
9 If an error in building a trace record occurred and DEBUG=NO is in effect, GTF processes as follows:	Event handler's error routine	

- Recoverable error — the record build routine puts an error indicator in the record field that caused the problem and places an address in the SDWA (via the SETRP macro) to point to the place where processing will resume.
- Unrecoverable error — GTF disables the event handler by issuing the SETEVENT macro instruction specifying DISABLE as the action. A SVC Dump is requested. Finally, message AHLI181 is issued specifying the event that is disabled. GTF then returns control to the MC routing facility to wait for other trace events. However, if all events are disabled, GTF returns control to RTM to terminate GTF.

Note: When GTF returns control to RTM for termination, percolation to the MC routing facility's error recovery routine occurs. This routine schedules on SRB which posts the enabled portion of GTF (the task monitor) for termination. See MO 8 for details of termination processing.

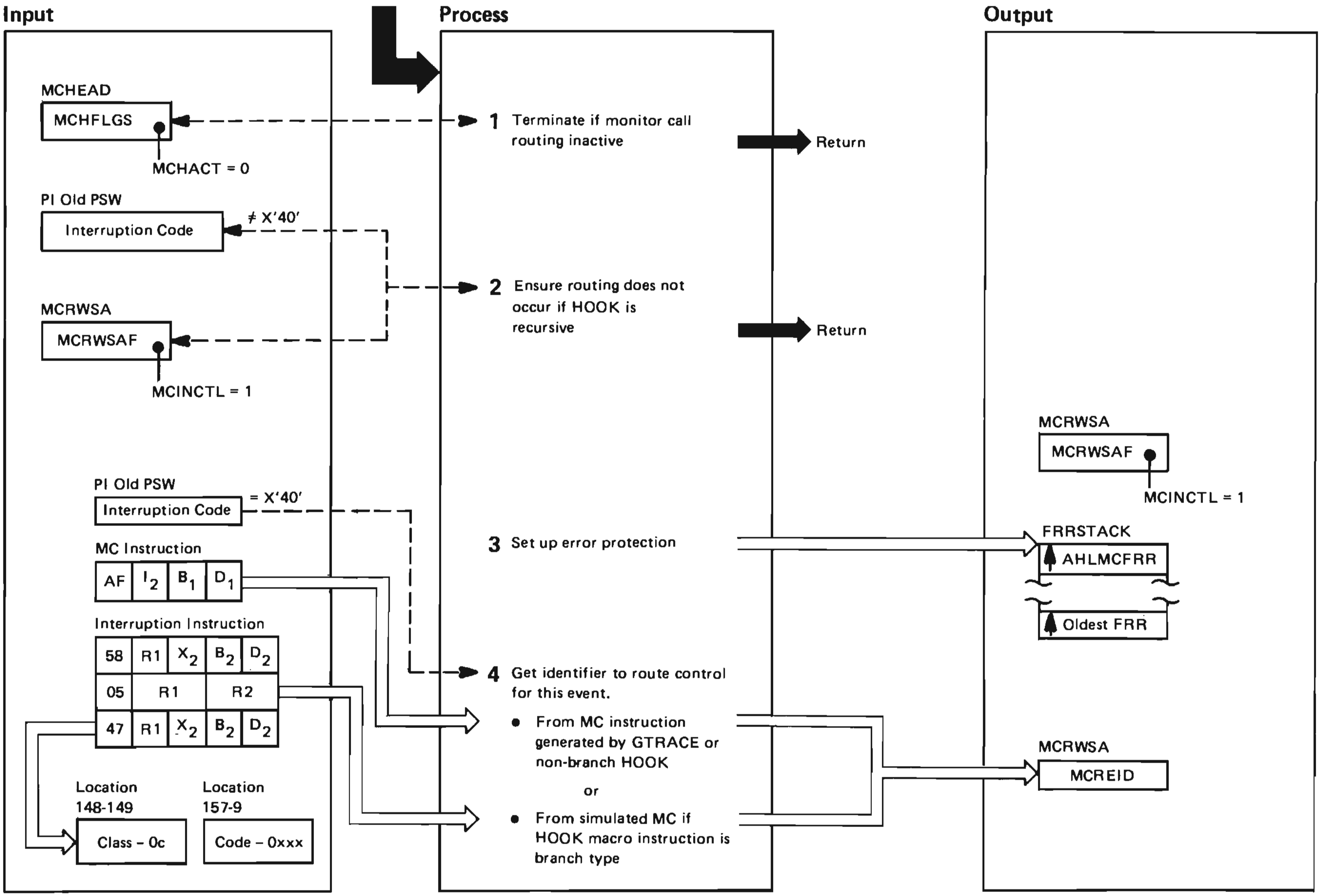
GTF Diagram 8. GTF Termination (Part 1 of 2)



GTF Diagram 8. GTF Termination (Part 2 of 2)

Extended Description	Module	Label
1 Before terminating, the writer subtask completes I/O for any filled buffers. GTF also completes any in-process transfer of data for ABEND/SNAP or SVC Dump. The asynchronous WTO subtask also completes its processing.	AHLCWRIT AHLIWRIT AHLWWRIT	
2 GTF issues the SETEVENT macro instruction specifying the FREE function to free GTF's MCQE and the event and class directories.	AHLTMON	
3 The record-blocking areas, buffers, and buffer control blocks are freed. The SRB used to schedule buffering is freed. Then, GTF frees the writer and WTO subtask by issuing the DETACH macro instruction.	AHLCWRIT AHLIWRIT AHLWWRIT	
Then it frees the message table (WMSGTAB) and the SRB it used for scheduling.	AHLWTASK	
4 GTF uses the GTFPCT pointers to the event handling routines to unfix the storage required for the routines. GTF then frees the storage required by GTFPCT.	AHLTMON	
5 Finally, GTF issues a DEQ macro instruction for the GTF resource and returns to the system.		

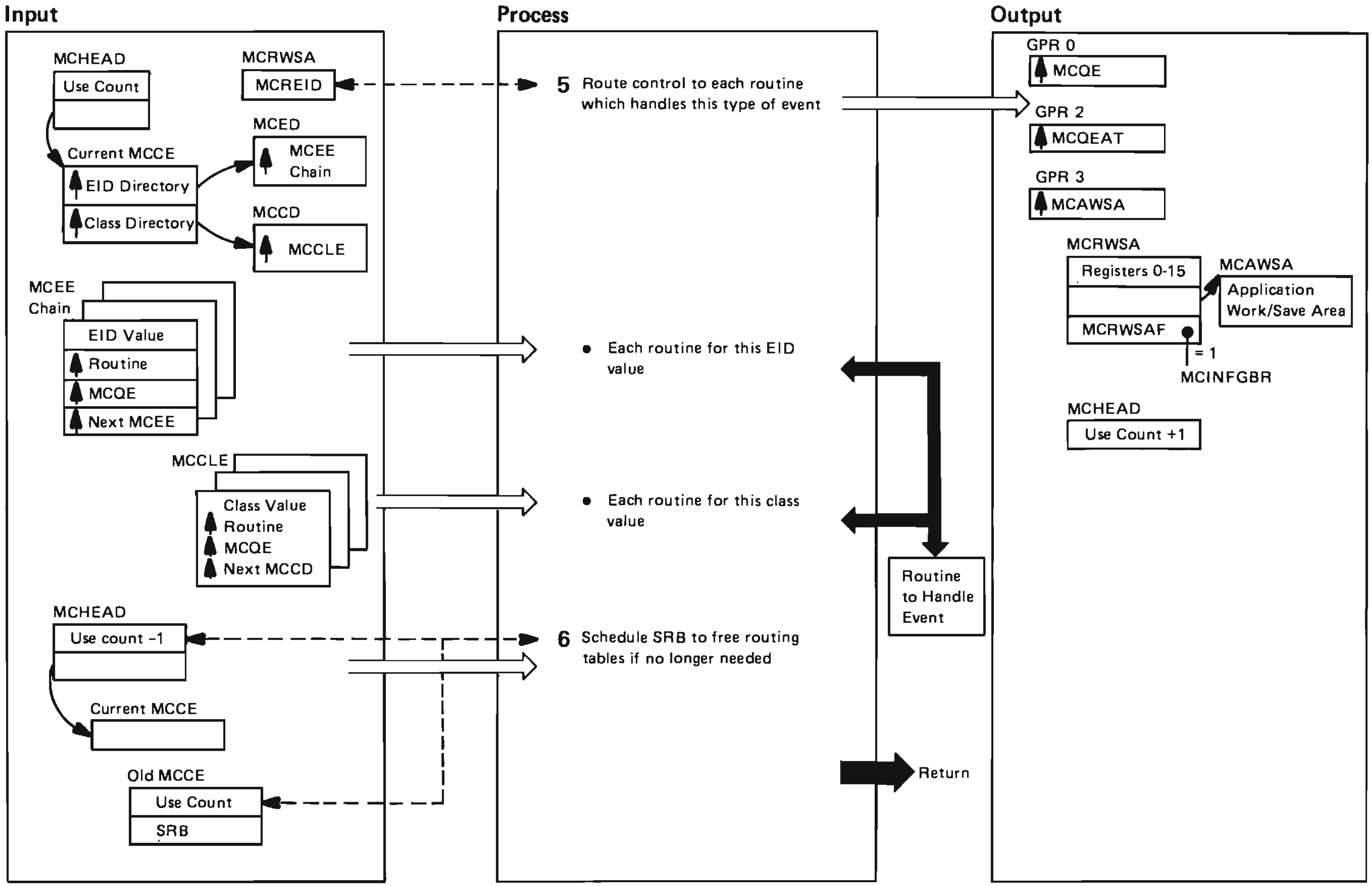
GTF Diagram 9. MC Event Handling (Part 1 of 4)



GTF Diagram 9. MC Event Handling (Part 2 of 4)

Extended Description	Module	Label
The MC interruption handling routine (AHLMCIH) receives control to trace events from the program check first-level interruption handler (PFLIH).		
1 The MC interruption handling routine determines whether routine is active by checking the MCHACT flag in MCHEAD. (The MCHEAD is described in *System Data Areas.*) Unless the flag has been set to 1 by the SETEVENT service, control is immediately returned to the caller.	AHLMCIH	
2 If the MCINCTL flag in the work/save area MCRWSA is on, recursion through AHLMCIH has occurred. In this case, control is returned to the PFLIH.	AHLMCIH	
3 The MC interruption handling routine sets the MCINCTL recursion flag in the MCRWSA. It also places the address of its error recovery routine on the FRR stack via SETFRR.	AHLMCIH	
4 The two-byte EID of the event to be routed is calculated as follows:	AHLMCER	
• MC instruction issued – the one-digit class and three-digit code from the instruction are taken from the MC hardware locations and combined to form the EID for the hook. The EID is then placed in the MCREID field of the MCRWSA.		
• Branch-HOOK macro issued - AHLMCIH simulated the MC instruction by obtaining the class and code from the NOP instruction following the BALR to AHLMCIH. It placed the values in low storage as if a monitor call had been issued. The EID value is calculated and placed in the MCRWSA as above.	AHLMCER	

GTF Diagram 9. MC Event Handling (Part 3 of 4)



Extended Description	Module	Label
5 The event router issues a CMSET for the home (PSAAOLD) address space for any non-PFLIH entries to GTF or for a PFLIH entry when PASID is not equal to HASID. The event router determines the routines to receive control for the event by using the EID directory (MCED) and the class directory (MCCD). The MC control element (MCCE) contains pointers to both directories. Before using the MCCE and directories, AHLMCER increases the use count in MCHEAD so these tables will not be released while the event is being handled.	AHLMCER	

The MCED is searched first for all routines to handle the event. ALHMCER uses the EID value from MCRWSA in the search. If an MCEE (event element) specifying the EID is not found, the class directory (CCD) is searched for routines to handle the event class. The class (calculated from the EID in the MCRWSA) is used as an index into the directory.

If jobname and/or address filtering need to be done, AHLTSELF is called. If the ASID or jobname is not selected for tracing, event routing is terminated. The event router passes the following information to the routine to build the record:

- Address of the MC queue element associated with the routine. (There is one MCQE for each MC user, for example, GTF.)
- Address of a work/save area (MCAWSA) for the event handler.
- Address of MCQEAT (for GTF, this is the address of the GTFPCT).

After the routine has completed processing, the event router resets the MCINFGBR to 0. The EID link field (MCEEEIDL) is checked to determine whether another MCEE or MCCLE exists for this event. If so, control is passed to the indicated routine as above.

Extended Description	Module	Label
6 The event router issues a CMSET to reset to the caller's environment if necessary. The event router attempts to decrease the use count in MCHEAD using a CDS instruction. If the attempt is unsuccessful, AHLMCER determines whether the pointer to the MCCE in MCHEAD has been changed. If it has, a new set of routing tables are in effect. In this case, the old use count from MCHEAD has been saved in the old MCCE. AHLMCER decreases this count, and if the result is 0, the tables associated with the old MCCE may be freed. This is done by scheduling the already initialized SRB located in the MCCE.	AHLMCER	
The event router resets the recursion flag (MCINTL) and returns to the caller of AHLMCIH. If an error occurs in an event handler and the event handler does not recover, the associated MCQE for the MC user is disabled, and that application is terminated. Other MC users still receive control for their specified events. If an error occurs while the event router is in control, the event router is terminated and message AHL132I is issued. All MCQEs are terminated in this case.	AHLMCER.	

Section 3: Program Organization

This section contains the following items which describe the organization of GTF.

- A description of GTF's use of storage during trace processing (see Figure 1-2).
- A list of the modules that are loaded for each trace option specified.
- Module calling sequences for GTF functions.
- A list of GTF load modules and the object modules they contain.
- A cross-reference table of module and data areas.

SQA	MCRWSA MCAWSA GTFBLOKs MC tables	
Pageable link pack area	data gathering routines£ AHLSBLOK[1] AHLSBUF AHLMCER[1] SETEVENT service	
GTF address space - private area	AHLTMON AHLxWRIT AHLWTASK	 x = C, I, or W
Subpool 6	GTFBCBs[1]	
Subpool 5	GTFBUFRx	
Nucleus	AHLMCIH[2] MCHEAD[2]	

[1] Fixed while GTF is active
[2] Always fixed

Figure 1-2. GTF Storage During Tracing

Trace Modules Loaded for Each Selected Option

Trace Option	Trace Modules
ASIDP	None
CCW, CCWP	AHLTCCWG
CSCH	AHLTFCG
DSP without SYSM	AHLTPID
DSP with SYSM	AHLTSYSM, AHLTXSYS
EXT	AHLTEXT
HSCH	AHLTFCG
IO	AHLTFCG
IOP	AHLTFCG, AHLTSYFL
JOBNAMEP	None
MSCH	AHLTFCG
PCI	None
PI	AHLTPID
PIP	AHLTPID, AHLTSYFL
RNIO without SYSM	AHLTFOR
RNIO with SYSM	AHLTSYSM, AHLTXSYS
RR	AHLTFOR
SLIP	AHLTSLIP
SRM without SYSM	AHLTFOR
SRM with SYSM	AHLTSYSM, AHLTXSYS
SSCH	AHLTFCG
SSCHP	AHLTFCG, AHLTSFL
SVC	AHLTSVC
SVCP	AHLTSVC, AHLTSYFL
SYS	AHLTFCG, AHLTSVC, AHLTPID, AHLTFOR, AHLTEXT
SYSM	AHLTSYSM, AHLTFCG
SYSP	AHLTFCG, AHLTSVC, AHLTPID, AHLTFOR, AHLTEXT, AHLTSYFL
TRC	None
USR, USRP	AHLTUSR

Figure 1-3. Trace Modules Loaded for Each Selected Option

Module Calling Sequences for GTF Functions

The following maps show sequential flow of GTF modules. Each map indicates the active modules for a specific function and describes the operations performed by those modules. Entry point names are also provided where they may differ from the module names. Figure 1-4 explains the design of sequence maps.

For more detailed descriptions of the functions, see the diagrams in Section 2: Method of Operation. A reference to the corresponding diagram accompanies each map.

Single Path Flow

Calling Module
- Module A
  - Module A^1
  - Module A^2
- Module B (ENTRY1)

The calling module first calls A, which then calls A^1 and A^2. When A returns, the caller passes control to B at entry point ENTRY1.

Alternate Path Flow

Calling Module
- Module A
- Module B
  - Module B^1

The calling module determines whether to call A or B for this operation. If B receives control, it calls B^1 before returning to the caller.

Figure 1-4. Example of Calling Sequence Map

GTF Control Flow

GTF Task Processing

Initialization - Includes initialization for GTF task and for GTF trace processing.
- Task monitor - Acknowledges STOP on termination ECBs while GTF is active.

Hook Processing

MC event-routing facility - Gives control to GTF when a trace event occurs.
- Filter/gather/build routines - Build GFT records for selected events. Request record blocking/buffering via AHLSTACK macro instruction.
  - Record-blocking routine - Stacks trace records into blocking area. Issues SCHEDULE macro instruction for buffer-copying routine when blocking area is full.

Buffer Processing

Asynchronous buffering routine - Copies contents of full blocking area to buffer. Issues POST macro instruction for GFT writer to manage buffer queues.

POST
- Writer routine - Manages buffer queues. Writes trace data to output data set (external trace mode only).

Initialization (Diagram 2)

- AHLGTFI - Enqueues GTF resource, sets up ESTAE for initialization, requests MC support, obtains and initializes GTFPCT, AHLPCITB, and INITDATA. Transfers control to AHLTMON after initialization.
  - AGLSCAN - Obtains GTF options from START command on EXEC statement or uses defaults. Sets options bits in GTFPCT indicating the START command options.
    - AHLTCTL1 - Controls obtaining and scanning of trace options.
      - AHLTSCN - Scans options from SYS1.PARMLIB or console. Sets GTFOPIND of INITDATA. Resolves conflicts of mutually exclusive options specified.
      - AHLTPMT - Builds and issues message AHLT1031.
      - AHLT103 - Obtains selective trace events from SYS1.PARMLIB or the console. Builds EBCDIC device tables for filtering.
      - AHLTTAB - Converts EBCDIC device tables to filter tables (masks).
- LOAD
  - Load appropriate record-building routines.
- ATTACH
  - { AHLIWRIT, AHLWWRIT, AHLCWRIT } Initializes blocking areas and buffers, opens DCB if external trace, completes ECBLIST, gets SRB to schedule buffering, loads and fixes AHLSBLOK and AHLSBUF.
- ATTACH
  - AHLWTASK - Sets up the message table (WMSGTAB) and an SRB to schedule WTO, builds ECBLIST.
- XCTL
  - AHLTMON - Performs final initialization; issues SETEVENT to: enable MC support, get CPU-related work areas (chained from LCCA), and initialize control register 8 on each active CPU. Waits on the ECB list. Posts writer and WTO ECBs for termination processing. Detaches writer and WTO subtasks. Cancels MC support.

SVC, EXT, RR, PI Minimal Record Building (Diagram 3)

- AHLMCER
  - AHLTSELF - Determines if an event occurred within a specified address space or job.
  - AHLTSYSM { AHLSVC, AHLEXT, AHLPI, AHLFRR, AHLSTAE } Builds a minimal record in MCAWSA for the event; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
    - record-blocking routine (via AHLSTACK macro instruction)

RNIO, SRM, DSP Minimal Record Building (Diagram 3)

AHLMCER - Determines that an event handled by AHLTXSYS has occurred.

- AHLTSELF - Determines if an event occurred within a specified address space or job.
- AHLTXSYS (AHLDSP, AHLSRB, AHLSRM, AHLRNIO) - Builds a minimal record in MCAWSA; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
  - record-blocking routine (via AHLSTACK macro instruction)

DSP, PI Comprehensive Record Building (Diagram 3)

AHLMCER - Uses hook EID and the event/class directories (MCED and MCCD) to determine that an event handled by AHLTSYFL or AHLTPID has occurred.

- AHLTSELF - Determines if an event occurred within a specified address space or job.
- AHLTSYFL - If selective tracing for PI is in effect, determines whether this event is selected by using the filter tables in GTFPCT.
- AHLTPID (AHLTDSP, AHLTLSR, AHLTPI, AHLTSRB) - Builds a comprehensive record in MCAWSA for the event; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
  - record-blocking routine (via AHLSTACK macro instruction)

RNIO, RR, SRM Comprehensive Record Building (Diagram 3)

AHLMCER - Determines that an event handled by AHLTFOR has occurred.

- AHLTSELF - Determines if an event occurred within a specified address space or job.
- AHLTFOR (AHLTFRR, AHLTSRM, AHLTSTAE, AHLTRNIO) - Builds a comprehensive record in MCAWSA for the event; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
  - record-blocking routine (via AHLSTACK macro instruction)

EXT Comprehensive Record Building (Diagram 3)

AHLMCER - Determines that an event handled by AHLTEXT has occurred.

- AHLTSELF - Determines if an event occurred within a specified address space or job.
- AHLTEXT - Builds a comprehensive record in MCAWSA for the event; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
  - record-blocking routine (via AHLSTACK macro instruction)

I/O, SSCH, CSCH, HSCH, MSCH, RSCH Record Building (Diagram 3)

- AHLMCER - Determines that an event handled by AHLTFCG, AHLTSYFL, or AHLTCCWG has occurred.
  - AHLTSELF - Determines if an event occurred within a specified address space or job.
  - AHLTSYFL - If selective tracing for I/O and/or SSCH is in effect, determines whether this event is selected by using the filter tables in GTFPCT.
    - AHLTFCG { AHLTFCG, RECSTACK } Builds a comprehensive record in MCAWSA for the event; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
      - record-blocking routine (via AHLSTACK macro instruction)
    - AHLTCCWG - Accesses the proper address spaces through a CMSET; traces the channel program; builds a trace record in the MCAWSA for the event, issues an AHLSTACK macro instruction to prefix the record and to cause record blocking.
      - record-blocking routine (via AHLSTACK macro instruction)

SVC Comprehensive Record Building (Diagram 3)

- AHLMCER - Determines that an event handled by AHLTSYFL or AHLTSVC has occurred.
  - AHLTSELF - Determines if an event occurred within a specified address space or job.
  - AHLTSYFL - If selective tracing for SVC is in effect, determines whether this event is selected. If so, passes control to AHLTSVC.
    - AHLTSVC Builds a comprehensive SVC record; issues AHLSTACK macro instruction to prefix the record and to cause record blocking.
      - record-blocking routine (via AHLSTACK macro instruction)

User/Component Record Building (Diagram 3)

- AHLMCER - Determines that an event handled by AHLTUSR has occurred.
  - AHLTSELF - Determines if an event occurred within a specified address space or job.
  - AHLTUSR (AHLTUSR) - Builds a record of data received from a user/component after checking validity of the data. Issues AHLSTACK macro instruction.
    - (AHLTUSRE) - If a page fault occurred in gathering data, indicates the condition to be handled by code generated by GTRACE macro instruction.
    - record-blocking routine (via AHLSTACK macro instruction)

Save Hook Control Record Building (Diagram 3)

- AHLMCER - Determines that an event handled by AHLTDIR has occurred.
  - AHLTSELF - Determines if an event occurred within a specified address space or job.
  - AHLTDIR - Builds a save hook control record for trace data requested by ABEND/SNAP/SVC Dump. Issues AHLSTACK macro instruction.
    - record-blocking routine (via AHLSTACK macro instruction)

SLIP Trace Record Building (Diagram 3)

```
AHLMCER           - Determines that an event handled by AHLTSLIP has occurred.
 |
 |______ AHLTSLIP { AHLSLSTD } Builds a trace record in MCAWSA for the event; issues
            |     { AHLSLUSR } AHLSTACK macro instruction to prefix the record and to
            |                  cause record blocking.
            |
            |____ record-blocking routine (via AHLSTACK macro instruction)
```

Record Blocking and Buffering (Diagram 4)

```
Record build routine -        Builds trace record in MCAWSA, then issues AHLSTACK macro
 |                            instruction which causes record blocking and buffering.
 |
 |______ AHLSBLOK (AHLSBLOK) - Places trace record with prefix in a blocking area
            |                  (GTFBLOK) in SQA.
            |
            |      (AHLSFEOB) - Swaps to an empty GTFBLOK when a blocking area
            |                  becomes full; keeps track of events lost while waiting for a
            |                  GTFBLOK. Schedules SRB for the buffering routine
            |                  (AHLSBUF) when a GTFBLOK is full or a hook from
            |                  ABEND/SNAP/SVC Dump is received.
            |
         SCHEDULE
            |
            |_____ AHLSBUF -  Moves blocked trace data to buffer (GTFBUFR); manages buffer
                              queues through buffer control blocks (GTFBCBs). Builds save list
                              element (SLE) if the blocked data contains a save-hook control
                              record for ABEND/SNAP/SVC Dump. Post writer.
```

GTF Writer (Diagram 5)

```
AHLSBUF           - Post GTF writer after copying a GTFBLOK to a buffer.
 |
POST                Note: only one writer module is loaded per GTF trace execution.
 |
 |    ____ AHLCWRIT - Tape writer; writes buffers from the write queue to output tape; maintains
 |   |                internal history queue.
 |   |
 |___|____ AHLIWRIT - Internal mode writer; maintains internal history queue of trace data.
     |
     |____ AHLWWRIT - Direct-access writer: writes buffers from the write queue to direct-access data
                      set; maintains internal history queue.
```


GTF - Dump Interfaces (Diagram 6)

SVC Dump - Requests buffers with data related to ASID of failing address space.

or

ABEND/SNAP (IGC0F05A) - Provides interface with GTF. Issues AHLREAD macro instruction which causes (via scheduling an SRB) trace data to be transferred from GTF's address space to SQA. Formats records. Calls for user format appendage or dumps user records in hexadecimal.

- AHLREADR is scheduled via AHLREAD macro instruction - Queues request for one page of trace data to be copied to SQA.
  - AHLxWRIT - Copies a page of data to SQA. Schedules SRB to return to AHLREADR to post the dump program. Before filling the last request associated with a dump, the writer removes BCBs from the SLE.

WTO for Disabled Event (Diagram 7)

Filter/gather/build error-recovery routine - If a disabling error occurs, calls for disabled event message.

- AHLWTO (AHLWTOMD) - Fills in table for message AHL118I informing operator of disabled event. Schedules SRB for WTO subtask.

  SCHEDULE

  - AHLWTASK (AHLWPOST) - Posts AHLWTASKS's ECB to write the message.
    (AHLWTASK) - Writes error message AHL118I.

Termination Processing (Diagram 8)

AHLTMON - Controls termination of GTF when STOP ECB or abnormal-end ECB is posted. Issues SETEVENT to free MC routing tables. Notifies writer and WTO subtasks of termination. Detaches subtask. Frees fixed storage. AHLTMON issues DEQ to free GTF resource.

- AHLxWRIT - Ensures all I/O to external data set is complete before freeing buffers, blocking areas, and related control blocks. Deletes AHLSBLOK and AHLSBUF.
- AHLWTASK - Frees message table area.

MC Event Handling (Diagram 9)

Program first level
Interruption handler - Passes control to the MC interruption handler when MC interruptions or other checks occur.

- AHLMCIH { AHLMCIH, AHLMCIHB, AHLMCIHC, AHLSTCLS, AHLVCCR8, AHLUTEST }
  - AHLMCER - calculates EID value for the hook and uses the event directory (MCED) and class directory (MCCD) to route control to each routine which handles this event type.
    - record-building routine

Section 4: Data Areas

This section contains descriptions of the data areas built and used by GTF and by the MC event handler. Refer to Data Areas for detailed formats of the data areas. This section also contains formats of the records built by GTF during tracing.

The GTF tables described are as follows:

- Buffer control block (GTFBCB).
- Initialization data area (INITDATA).
- Primary control table (GTFPCT).
- Range table (RANGETAB).
- Save list element (SLE).
- CCW trace work area (AHLCCWWK).
- PCI work area (AHLPCITB).

The MC event handler tables are as follows:

- Application work/save area (MCAWSA).
- Base table (MCHEAD).
- Class directory (MCCD).
- Class element (MCCLE).
- Control element (MCCE).
- Queue element (MCQE).
- Router work/save area (MCRWSA).

The GTF record formats are as follows:

- Control records (lost block, lost event, save hook, and timestamp).
- Minimal trace records (DSP, EXT, PI, RNIO, RR, SRM, SVC).
- Comprehensive trace records (DSP, EXT, PI, RNIO, RR, SRM, SVC).
- Trace records (I/O, CSCH, HSCH, MSCH, RSCH, SLIP, SSCH).

GTF Data Areas

Buffer Control Block (GTFBCB)

Created by: AHLxWRIT (x is C, I or W depending on the GTF mode specification and/or output device).

Updated by: AHLSBUF.

Use: A temporary data area that contains indicators of the GTF options obtained during initialization. The table is freed at the end of initialization.

PCI Work Area (AHLPCITB)

Created by: AHLGTFI.

Updated by: AHLTCCWG.

Use: A table that contains command codes, flags, and channel program starting addresses for PCI interrupts.

Primary Control Table (GTFPCT)

Created by: AHLGTFI.

Updated by: AHLxWRIT, AHLREADR, AHLSBLOK, AHLSBUF, AHLSCAN, AHLTCTL1, AHLTEXT, AHLTFOR, AHLTMON, AHLTPID, AHLTPMT, AHLTSCN, AHLTFCG, AHLTSVC, AHLTSYFL, AHLTSYSM, AHLTTAB, AHLTUSR, AHLTXSYS, AHLWTASK, AHLTSLIP.

Referenced by: AHLREADR, AHLSTACK, AHLT103, AHLWTO, IGC0F05A.

Use: Contains the majority of control information required for GTF operation. This includes pointers to buffer queues, indicators of options in effect, pointers to SRBs for scheduling services, pointers to the filter tables.

Range Table (RANGETAB)

Created by: AHLGTFI.

Updated by: AHLGTFI.

Use: A temporary data area for GTF initialization that contains starting and ending addresses of each event-handler routine required for tracing. From this table, the range of storage required for page fixing is determined. The table is freed at the end of initialization.

Save List Element (SLE)

Created by: AHLxWRIT (x is C, I or W depending on the GTF mode and/or output device).

Updated by: AHLSBUF.

Use: Contains the control information related to a request by ABDUMP/SNAP or SVC Dump for trace data. This information includes the TCB address for a dumped task, the ASID of a dumped address space and pointers to the buffers containing the required data and/or related buffer control blocks.

MC Event Handling/Data Areas

Application WCCk/Save Area (MCAWSA)

Created by: AHLSETEV, AHLVCON.

Updated by: AHLMCER, AHLSBLOK, AHLSETD, AHLSTACK, AHLTDIR, AHLTEXT, AHLTFOR, AHLTPID, AHLTFCG, AHLTSVC, AHLTSYSM, AHLTUSR, AHLTXSYS, AHLVCOFF, AHLWTO, AHLTSLIP, AHLMCIH.

Use: Used by the data-gathering routines for register save areas, temporary work space, and record building. Records are moved, as they are completed, from this data area to the blocking areas (GTFBLOKs).

Base Table (MCHEAD)

Created by: Assembled as part of the MC interruption-handler routine (AHLMCIH), which is resident in the nucleus.

Updated by: AHLSETEV, IEAVNP17.

Referenced by: AHLMCER, AHLREADR, AHLSETD, AHLVCON, IGC0F05A.

Use: Contains basic control information for the MC event handler. The information includes the address of the current (active) control element (MCCE), a use count indicating the number of routines using this set of tables, and the current active class mask in control register eight.

CCW Trace Work Area (AHLCCWWK)

Created by: AHLSETEV

Updated by: AHLTCCWG

Use: A temporary work area for CCW trace that contains flags, bit switches, addresses, and counters. One work area exists for each CPU that is in a multiprocessing configuration and that is running CCW trace.

Class Directory (MCCD)

Created by: AHLSETEV.

Referenced by: AHLMCER.

Use: Contains the addresses of MC class elements (MCCLEs); used as an index to route control of trace events by class.

Class Element (MCCLE)

Created by: AHLSETEV.

Referenced by: AHLMCER, AHLSETD.

Use: Specifies a class of events to be traced and the event-handler routine to receive control for that class.

Control Element (MCCE)

Created by: AHLSETEV.

Updated by: AHLMCER, AHLSETD.

Referenced by: AHLREADR, AHLSET, IGC0F05A.

Use: Contains the addresses of the routine directories, the MC class directory (MCCD), and the MC event directory (MCED). Also contains the address of the first application control table (MCQE) and the SRB used to free the tables associated with this MCCE.

Queue Element (MCQE)

Created by: AHLSETEV.

Updated by: AHLGTFI (only the MCQE for the GTF application).

Referenced by: AHLxWRIT, AHLMCER, AHLREADR, AHLSETD, AHLSTACK, AHLTMON.

Use: Contains control information to define an MC application, for example, GTF. It contains the application's name, the address of the next MCQE, an ECB used to terminate the application, and the addresses of the chains of MC class elements (MCCLEs) and of MC event elements (MCEEs).

Event Handler Work/Save Area (MCRWSA)

Created by: IPL/NIP, VARY CPU ONLINE.

Updated by: AHLMCER, AHLMCIH, AHLSETD, AHLSETEV, AHLVCOFF, AHLVCON, IEAVNP17.

Referenced by: AHLTEXT, AHLTFOR, AHLTPID, AHLTFCG, AHLTSVC, AHLTSYFL, AHLTUSR, AHLWTO, AHLTSLIP.

Use: Contains register save area for the MC interrupt and the address of the MCAWSA. It is located in SQA and is pointed to by the LCCA work/save area vector table.

GTF Control Records

Lost Block Record

GTF places a lost block control record in the trace buffers whenever data contained in a GTFBLOK has been lost.

Offset		Size	Description
0	(0)	2	Record length in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'00' indicates control record.
5	(5)	1	Format identifier: X'03'.
			X'05' for buffer full condition;
6	(6)	4	Time zone value.
10	(A)	8	TOD clock value.
18	(12)	4	Lost event count.

Lost Event Record

GTF places a lost data control record in the trace buffers whenever data about an interruption is missed because buffers were full.

Offset		Size	Description
0	(0)	2	Record length in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'00' indicates control record.
5	(5)	1	Format identifier: X'02'.
			X'05' for buffer full condition;
6	(6)	4	Time zone value.
10	(A)	8	TOD clock value.
18	(12)	4	Lost event count.

Save Hook Control Record

GTF places a save hook control record in the trace buffers whenever a request from ABEND/SVC Dump is received for trace data.

Offset		Size	Description
0	(0)	2	Record length in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'00' indicates control record.
5	(5)	1	Format identifier: X'00'.
			X'05' for buffer full condition;
6	(6)	4	Time zone value.
10	(A)	8	TOD clock value.
18	(12)	2	EID, F200, F100
20	(14)	2	ASID
22	(16)	4	TCB

Time Stamp Control Record

The first record in every variable block.

Offset		Size		Description
0	(0)	2		Record length in bytes.
2	(2)	2		X'0000'.
4	(4)	1		Record identifier: X'00' means this is a control record.
5	(5)	1		Format identifier: X'01'.
6	(6)	4		Time zone value.
10	(A)	8		TOD clock value.
18	(12)	8		GTF option bytes.
18	(12)	1		Option byte 1:
		1....	....	SYSM in effect.
		.1..	....	SYSP in effect.
		..1.	....	SYS in effect.
		...1	....	USR in effect.
		....	1...	TRC in effect.
		....	.1..	DSP in effect.
		....	..x.	Reserved
		....	...1	PCI in effect.
19	(13)	1		Option byte 2:
		1...	....	SVC in effect.
		.1..	....	SVCP in effect.
		..1.	....	SIO specified; SSCH in effect.
		...1	....	SIOP specified; SSCHP in effect.
		....	1...	PI in effect.
		....	.1..	PIP in effect.
		....	..1.	IO in effect.
		....	...1	IOP in effect.
20	(14)	1		Option byte 3:
		1...	....	EXT in effect.
		.1..	....	RNIO in effect.
		..1.	....	SRM in effect.
		...1	....	RR in effect.
		....	1...	SLIP in effect.
		....	.1..	CCW in effect.
		....	..1.	CCWP in effect.
		....	...1	IO = SIO: the devices traced selectively for IO and SIO are identical.
21	(15)	1		Option byte 4:
		1...	....	CCW=I in effect.
		.1..	....	CCW=S in effect.
		..1.	....	JOBNAMEP in effect.
		...1	....	ASIDP in effect.
		....	1...	USRP in effect.
		....	.xx.	Reserved.
		....	...1	User timestamp requested in each trace record.
22	(16)	1		Option byte 5:
		1...	....	SSCH in effect.
		.1..	....	SSCHP in effect.
		..1.	....	MSCH in effect.
		...1	....	HSCH in effect.
		....	1...	CSCH in effect.
		....	.xx.	Reserved.
		....	...1	IO = SSCH; the devices traced selectively for IO and SSCH are identical.
23	(17)	3		Reserved.

GTF Trace Records

CCW Trace Record

GTF builds a record that has the following format when TRACE=CCW or TRACE=CCWP is in effect, and either a start subchannel (SSCH) occurs in the I/O supervisor SSCH subroutine, or an I/O interruption occurs. To trace channel programs, TRACE=SSCH or TRACE=I/O must also be in effect.

Offset		Size	Description
0	(0)	2	Record length in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'07'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device address.
32	(20)	2	Maximum amount of data for each CCW.
34	(22)	1	Flags for CCW trace formatter.
35	(23)	2	Sequential record count for print dump of this event.
37	(25)	1	Translate table ID for formatter.
38	(26)	2	Flags describing data.
40	(28)	2	Length of data in this CCW.
42	(2A)	1	Length of data in this record.
43	(2B)	4	Virtual address of CCW.
47	(2F)	8	CCW being traced.
55	(37)	217	Data area or repeat of offset 37-55 if space available.

CCW Trace Message Record

GTF builds a record that has the following format when CCW trace has to output messages AHL140D, AHL141D, AHL146I, AHL147I, AHL148I, AHL149I, AHL150I, AHL151I, AHL152I, AHL153I, or AHL154I.

Offset		Size	Description
0	(0)	2	Record length in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'07'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device address.
32	(20)	2	Not used; contains value used in CCW trace record.
34	(22)	2	Flags for CCW trace formatter.
35	(23)	2	Sequential record count for print dump of this event.
37	(25)	4	'MSG' ID.
41	(29)	4	Not used; zeroes.
45	(2D)	2	Length of message.
47	(2F)	225	Message data.

CSCH/HSCH Trace Record

GTF builds a record that has the following format when either a CSCH or an HSCH event occurs and TRACE=SYSM, TRACE=SYS, TRACE=SYSP, TRACE=CSCH, or TRACE=HSCH is in effect. This record is common to both the CSCH event and the HSCH event. They can be differentiated by the EID value (X'5105' for CSCH, X'5103' for HSCH) found in the prefix section of the record at offset X'E'.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'00'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier. X'5102' indicates a CSCH event. X'5103' indicates a HSCH event.
16	(10)	2	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device number.
32	(20)	2	Startability flags.
34	(22)	4	Subchannel identifier.
38	(26)	1	Condition code.
39	(27)	1	Driver identifier.
40	(28)	1	Associated request driver identifier.
41	(29)	1	IOSLEVEL.
42	(2A)	1	UCBLEVEL.

DSP Comprehensive Trace Record

GTF builds a record that has the following format when an entry is made to the dispatcher to dispatch a unit of work and TRACE=DSP is the GTF option in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates a trace record.
5	(5)	1	Format identifier: X'02'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier. X'0001' indicates SRB dispatching. X'0002' indicates LSR dispatching. X'0003' indicates TCB dispatching. X'0004' indicates exit prolog dispatching.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	8	Resume PSW for new task.
38	(26)	4	New TCB (for LSR and TCB, SRB for SRB).
For TCB only:			
42	(2A)	8	CDE name.
For SRB only:			
42	(2A)	4	Parm address.
46	(2E)	1	Global or local.

DSP Minimal Trace Record

GTF builds a record with the following format when an entry is made to the dispatcher to dispatch a unit of work and both TRACE=SYSM,DSP are the GTF options in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03' indicates AMDSYS03.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock.
14	(E)	2	Event identifier. X'0001' indicates SRB dispatching. X'0002' indicates LSR dispatching. X'0003' indicates TCB dispatching. X'0004' indicates exit prolog dispatching.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Resume PSW for work unit being dispatched.
30	(1E)	4	Current TCB or N/A (for TCB and LSR only).
34	(22)	4	Register 15.
38	(26)	4	Register 0 or SRB.
42	(2A)	4	Register 1.
46	(2E)	1	For SRB only; global or local.

EWA Trace Record

GTF builds a record that has the following format if an EWA is present when CCW=IOSB is specified with TRACE=CCW or TRACE=CCWP.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'07'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device address.
32	(20)	2	Not used; contains value used in CCW trace record.
34	(22)	1	Flags for CCW trace formatter.
35	(23)	2	Sequential record count for print dump of this event.
37	(25)	4	'EWA' ID.
41	(29)	4	Address of EWA.
45	(20)	2	Length of EWA in this record.
47	(2F)	225	EWA data area.

EXT Comprehensive Trace Record

GTF builds a record that has the following format when an external interruption occurs and either TRACE=SYS or TRACE=EXT is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates a trace record.
5	(5)	1	Format identifier: X'02'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'6201' indicates an external interruption.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	8	External old PSW.
38	(26)	4	Old TCB.
For SIGP interrupt:			
42	(2A)	4	PARMFIELD.
46	(2E)	2	CPUID.
For clock comparator interrupt.			
42	(2A)	2	TQE flags.
44	(2C)	4	TQE exit.
48	(30)	4	TQE ASCB.
For CPU timer interrupt:			
42	(2A)	2	TQE flags.
44	(2C)	4	TQE exit.

EXT Minimal Trace Record

GTF builds a record with the following format when an external interruption occurs and TRACE=SYSM is the GTF option in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'6201' indicates an external interruption.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	External old PSW.
30	(1E)	4	TCB of interrupted task or N/A.
34	(22)	4	NTQE TCB or INT CPU or N/A.

IOSB Trace Record

GTF builds a record that has the following format; GTF builds the record either when CCW=ISOB is specified with TRACE=CCW or TRACE=CCWP, or when an error message is outputted while tracing a channel program.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'07'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device address.
32	(20)	2	Not used; contains value used in CCW trace record.
34	(22)	1	Flags for CCW trace formatter.
35	(23)	2	Sequential record count for print dump of this event.
37	(25)	4	'IOSB' ID.
41	(29)	4	Address of IOSB.
45	(2D)	2	Length of IOSB.
47	(2F)	225	IOSB data area.

I/O Trace Record

GTF builds record that has the following format when an I/O interruption occurs and TRACE=SYSM, TRACE=SYS, TRACE=IO, or TRACE=IOP is in effect. To trace PCI I/O interruptions, TRACE=PCI must also be in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'00'.
6	(6)	8	Optional timestamp; if TIMESTAMP=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'2100' indicates a PCI I/O interruption. X'5101' indicates an EOS I/O interruption. X'5200' indicates an I/O interruption with a valid UCB.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device number.
32	(20)	8	I/O old PSW.
40	(28)	16	IRB.
56	(38)	4	TCB.
60	(3C)	2	Sense bytes.
62	(3E)	2	Flags.
64	(40)	1	Driver identifier.
65	(41)	1	IOSLEVEL.
66	(42)	1	UCBLEVEL.

MSCH Trace Record

GTF builds a record that has the following format when an MSCH event occurs and TRACE=SYSM, TRACE=SYS, TRACE=SYSP, or TRACE=MSCH is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'00'.
6	(6)	8	Optional timestamp; if TIMESTAMP=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'5104' indicates an MSCH event.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device number.
34	(22)	4	Subchannel identifier.
35	(23)	1	Condition code.
36	(24)	1	IOSOPT.
37	(25)	1	IOSOPT2.
38	(26)	1	IOSLEVEL.
66	(42)	28	First SCHIB.
67	(43)	1	UCBLEVEL.
95	(5F)	28	Second SCHIB.

PI Comprehensive Trace Record

GTF builds a record that has the following format when a program interruption occurs and either TRACE=PI or TRACE=SYS in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'02'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'6101' indicates a program interruption with codes 1-17, 19, 128. X'6200' indicates a program interruption with code 18.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	8	Program old PSW.
38	(26)	4	INT TCB.
42	(2A)	4	Virtual page address.
46	(2E)	8	RB or CDE name.
54	(36)	64	Registers 0-15.

PI Minimal Trace Record

GTF builds a record that has the following format when a program interruption occurs and TRACE=SYSM is the GTF option in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'6101' indicates a program interruption with codes 1-17, 19, 128. X'6200' indicates a program interruption with code 18.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	8	Program old PSW.
30	(1A)	4	Old TCB.
34	(1E)	4	Virtual page address.
38	(26)	4	Register 15.
42	(2A)	4	Register 1.

RNIO Comprehensive Trace Record

GTF builds a record that has the following format when ACF/VTAM requests an I/O path information unit (PIU) and the GTF options in effect are MODE=EXT and TRACE=RNIO.

Note: ACF/VTAM Version 2 Diagnosis Guide describes the access method used to create the PIU data.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains 4 bytes of the CVTTZ field and the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'8200' indicates PIU output. X'8100' indicates PIU input.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	20	RNIO PIU data. Minimum of 10 bytes. Maximum of 20 bytes. The fifth halfword contains the number of remaining bytes. When PIU data is less than 20 bytes, the field is padded on the right with blanks.

RNIO Comprehensive Trace Record with ACF/VTAM

GTF builds a record that has the following format when ACF/VTAM requests an I/O path information unit (PIU) and the GTF options in effect are MODE=EXT and TRACE=RNIO.

Note: ACF/VTAM Version 2 Diagnosis Guide describes the access method used to create the PIU data.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'8200' indicates PIU output. X'8100' indicates PIU input.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	20	RNIO PIU data. Valid range is 1-20 bytes. R0 contains the number of bytes to collect. If R0 is greater than 20, 20 bytes are collected. If R0 is less than 20, the field is padded on the right with blanks.
50	(32)	4	Contents of Register 0.

RNIO Minimal Trace Record

GTF builds a record that has the following format when ACF/VTAM requests an I/O path information unit (PIU) and the GTF option in effect is either MODE=INT or the combination of TRACE=SYSM and TRACE=RNIO.

Note: ACF/VTAM Version 2 Diagnosis Guide describes the access method used to create the PIU data.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains 4 bytes of the CVTTZ field and the eight bytes of the TOD clock value. X'8200' indicates PIU output. X'8100' indicates PIU input.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	20	RNIO PIU data. Minimum of 10 bytes. Maximum of 20 bytes. The fifth halfword contains number of remaining bytes. When PIU data is less than 20 bytes, the field is padded on the right with blanks.

RNIO Minimal Trace Record with ACF/VTAM

GTF builds a record that has the following format when ACF/VTAM requests an I/O path information unit (PIU) and the GTF option in effect is either MODE=INT or the combination of TRACE=SYSM and TRACE=RNIO.

Note: *ACF/VTAM Version 2 Diagnosis Guide* describes the access method used to create the PIU data.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'8200' indicates PIU output. X'8100' indicates PIU input.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	20	RNIO PIU data. Valid range is 1-20 bytes. R0 contains the number of bytes to collect. If R0 is greater than 20, 20 bytes are collected. If R0 is less than 20, the field is padded on the right with blanks.
42	(2A)	4	Contents of R0.

RR Comprehensive Trace Record

GTF builds a record that has the following format when an invocation of a recovery routine occurs and TRACE=SYS or TRACE=RR is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04' indicates AMDSYS04.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock.
14	(E)	2	Event identifier. X'4002' indicates STAE/ESTAE invocation. X'4003' indicates FRR invocation.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	8	Name of recovery routine or U/A.
38	(26)	8	PSW current when error occurred.
46	(2E)	4	Completion code.
50	(32)	8	Error flags from system diagnostic work area (SDWA).
58	(3A)	4	Retry address or N/A.
62	(3E)	4	Address of SDWA (STAE/ESTAE only).

RR Minimal Trace Record

GTF builds a record that has the following format when an invocation of a recovery routine occurs and TRACE=SYS is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03' indicates AMDSYS03.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock.
14	(E)	2	Event identifier: X'4002' indicates STAE/ESTAE invocation. X'4003' indicates FRR invocation.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Error PSW.
30	(1E)	4	Completion code.
34	(22)	4	Error flags from system diagnostic work area (SDWA).
38	(26)	3	Additional error flags from SDWA.
41	(29)	1	Return code from recovery routine (STAE/ESTAE only).
42	(2A)	4	Retry address or N/A. **Note:** If no return code at offset 41, begin retry address at offset 41.
46	(2E)	4	Address of SDWA (STAE/ESTAE only). **Note:** If retry address begins at offset 41, SDWA address begins at offset 45.

RSCH Trace Record

GTF builds a record that has the following format when an RSCH event occurs and TRACE=SYSM, TRACE=SYS, TRACE=SYSP, or TRACE=RSCH is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'00'.
6	(6)	8	Optional timestamp; if TIMESTAMP=YES is specified, this field contains eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'5106' indicates an RSCH event.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device number.
32	(20)	4	Real address of channel program.
36	(24)	4	Virtual address of channel program.
40	(28)	4	IOSDSID.
44	(2C)	1	Condition code.
45	(2D)	8	Dynamic seek address.
53	(35)	1	IOSGPMSK.
54	(36)	1	IOSOPT.
55	(37)	1	IOSFMSK.
56	(38)	1	Driver identifier.
57	(39)	1	IOSLEVEL.
58	(3A)	1	UCBLEVEL.

SLIP Trace Records

GTF builds a SLIP trace record when TRACE=SLIP is the GTF option in effect and:

- A SLIP trap has matched and either TRACE or TRDUMP has been specified on the SLIP command.
- A SLIP trap is in DEBUG mode (specified on the SLIP command) and is inspected by the SLIP processor as a result of any SLIP event.

The SLIP trace records are:

- SLIP Standard Trace Record
- SLIP Standard/User Trace Record
- SLIP User Trace Record
- SLIP DEBUG Trace Record

SLIP Standard Trace Record: GTF builds a record that has the following format when the HOOK macro instruction is issued with an EID of IEAVSLSD.

Notes:

1. *U/A indicates the data is unavailable.*
2. *N/A indicates the data is not applicable.*
3. *Asterisks (** ... **) are used if an error occurred when attempting to obtain data or the data is unavailable because it is paged out.*

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04' indicates AMDSYS04.
6	(6)	8	Optional timestamp; if GTF option TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'4004' indicates a SLIP standard trace record.
16	(10)	4	ASCB address.
20	(14)	2	CPUID. (**Note:** When SLIP is entered from RTM2, the CPUID recorded may be different from the CPUID when RTM2 was running.
22	(16)	8	Jobname from current address space (or N/A).
30	(1E)	4	SLIP trap ID.
34	(22)	2	ASID of current address space.
36	(24)	8	Job step program name (or U/A or N/A).
44	(2C)	4	TCB address (or N/A).

Offset		Size		Description
48	(30)	1		System mode indicators, byte 1:
		1....	....	Supervisor control mode.
		.1..	....	Disabled for I/O and external interrupts.
		..1.	....	Global spin lock held.
		...1	....	Global suspend lock held.
		....	1...	Local lock held.
		....	.1	Type 1 SVC in control.
		....	..1.	SRB mode.
		....	...1	TCB mode.
49	(31)	1		System mode indicators, byte 2:
		1...	....	Recovery routine in control (always zero if a PER interrupt).
		.1..	....	Problem program state.
		..1.	....	Supervisor state.
		...1	....	System key.
		....	1...	Problem program key.
		....	.1..	Any global lock held.
		....	..1.	Any lock held.
50	(32)	1		Error byte 1 (or zeros if a PER interrupt):
		1...	....	Program check interrupt.
		.1..	....	Restart interrupt.
		..1.	....	SVC error.
		...1	....	Abend; task issued SVC 13.
		....	1...	Paging I/O error.
		....	.1..	Dynamic address translation error.
		....	..1.	Software error caused by machine check.
		....	...1	Abnormal address space termination.
51	(33)	1		Error byte 2 (or zeros if a PER interrupt):
		1...	....	Memterm.
52	(34)	1		SLIP flags:
		1...	....	DEBUG record.
		.1..	....	Registers collected.
53	(35)	2		Data unavailable counter (or zeros if DATA was not specified for the trap).

The following fields apply to PER interrupts only (otherwise set to N/A (or N or one-byte fields)).

Offset		Size		Description
55	(37)	8		Load module name in which the interrupt occurred (or U/A or N/A).
63	(3F)	4		Offset in load module (or U/A or N/A).
71	(47)	6		Instruction content (six bytes of data beginning at the address of the instruction that caused the PER interrupt).
77	(4D)	4		Target instruction address if EXECUTE instruction (or N/A or U/A).
81	(51)	66		Target instruction content if EXECUTE instruction (six bytes of data beginning at the target instruction address), or (N/A or U/A).
87	(57)	4		Beginning range virtual address if SA (storage-alteration) specified on SLIP command (or N/A).
91	(5B)	4		Four bytes of storage starting at beginning range virtual address if SA specified (or N/A or U/A).
95	(5F)	8		Program old PSW.
103	(67)	4		Program interrupt code (PIC) and instruction length code.
107	(68)	1		PER interrupt code:
		1..	....	Successful-branch event (SB).
		.1..	....	Instruction-fetch event (IF).
		..1.	....	Storage-alteration event (SA).

Offset		Size		Description
108	(6C)	1		PER trap mode:
		1...	....	Successful-branch monitoring (SB).
		.1..	....	Instruction-fetch monitoring (IF).
		..1.	....	Storage-alteration monitoring (SA).
		...x	....	Reserved.
		....	1...	PER trap.
		....	.1..	Recovery specified.
		....	..1.	Message flag.
		....	...1	Message flag.
109	(6D)	2		Keymask.
111	(6F)	2		SASID.
113	(71)	2		Authorization index.
115	(73)	2		PASID.
117	(75)	1		PSW S bit indicator (F1 indicates secondary addressing mode, F0 indicates primary addressing mode).
118	(76)	8		Reserved.

SLIP Standard + User Trace Record: GTF builds a record that has the following format when the HOOK macro instruction is issued with an EID of IEAVSLSU.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04' indicates AMDSYS04.
6	(6)	8	Optional timestamp; if GTF option TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'4005' indicates a SLIP standard/ user record.
16	(10)	4	
through			Fields are identical to the SLIP standard record.
118	(76)	18	
136	(88)	2	Length of user-defined data.
138	(8A)	variable	User-defined data (specified via the TRDATA parameter on the SLIP command).

SLIP User Trace Record: GTF builds a record that has the following format when the HOOK macro instruction is issued with an EID of IEAVSLUR.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04' indicates AMDSYS04.
6	(6)	8	Optional timestamp; if GTF option TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'4006' indicates a SLIP user record.
16	(10)	2	CUPUID (See Note 3).
18	(12)	2	Extension number (See Note 3).
20	(14)	1	Continuation length (See Note 3).
21	(15)	2	Length of the user defined data (See Notes 1, 2, and 3).
23	(17)	variable	User-defined data (specified via the TRDATA parameter on the SLIP command). (See Note 3).

Notes:

1. *If the SLIP user requests registers to be placed in the SLIP user record, they will be the first field in the record.*

2. *A length field of zero indicates that the user-defined data was not available (for example, the data is paged out).*

3. *The TRDATA parameter on the SLIP command specifies one or more data ranges. The number of records needed to trace these ranges depends on the size of the ranges specified. The trace contains a standard plus (+) user record from the next range or a user record followed by as many user records and user continuation records as needed to trace the ranges specified. The header for each record contains the CPUID and extension number to help correlate the output (extension numbers apply only to user and user continuation records). When a record is partially filled and the data from the next range will not fit in the remaining space; the AHLSTACK macro instruction writes the partially filled record to the trace data set. Another user record is built, the extension number is increased by one, and the continuation length is set to zero. The data length and data is then copied into this record.*

   When the length of the data from a range is greater than 249 bytes, the excess data is put in user continuation records in the following manner. The data length and first 248 bytes are put in a user record. After writing that record a user continuation record is built. The extension number is increased by one and the continuation length is set to the number of bytes of data to be put in this record. If more than 251 bytes of data are left, 248 bytes are copied into record, and it is written. User continuation records are built until all the data in from that range is traced.

SLIP DEBUG Trace Record: GTF builds a record that has the following format when the HOOK macro instruction is issued with an EID of IEAVSLSU and the SLIP trap is in DEBUG mode.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04' indicates AMDSYS04.
6	(6)	8	Optional timestamp; if GTF option TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'4005' indicates SLIP standard/ user trace record.
16	(10)	4	
through			Fields are identical to the SLIP standard record. (See Note 1.)
109	(6D)	27	

Offset		Size		Description	
136	(78)	1		DEBUG byte 1:	See Note 2.
		1....	....	ADDRESS test.	
		.1..	....	ASID test.	
		..1.	....	COMP test.	
		...1	....	DATA test.	
		....	1...	ERRTYP test.	
		....	.1..	JOBNAME test.	
		....	..1.	JSPGM test.	
		....	...1	LPAMOD test.	
137	(79)	1		DEBUG byte 2:	
		1...	....	MODE test.	
		.1..	....	PVTMOD test.	
		..1.	....	RANGE test.	
		...1	....	ASIDSA test.	

Notes:

1. *The high-order bit in the SLIP flags (SFLG) field (at offset X'34') is set on to indicate a DEBUG record.*

2. *For each SLIP event, these bytes (at offsets (X'78' and X'79') represent:*

   - *A bit set to one indicates that the corresponding keyword test did not match.*

   - *Bits set to zero indicate that the corresponding keyword test (1) matched, or (2) was not performed.*

SRM Comprehensive Trace Record

GTF builds a record that has the following format when an invocation of the system resource manager occurs and TRACE=SYS or TRACE=SRM is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'04' indicates AMDSYS04.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock.
14	(E)	2	Event identifier: X'4001'.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	4	Register 15.
34	(22)	4	Register 0.
38	(26)	4	Register 1.

SRM Minimal Trace Record

GTF builds a record that has t he following format when an invocation of the system resource manager occurs and TRACE=SYSM is the GTF option in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'03' indicates AMDSYS03.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'4001'.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	4	Register 15.
26	(1A)	4	Register 0.
30	(1E)	4	Register 1.

SSCH Trace Record

GTF builds a record that has the following format when an SSCH event occurs and TRACE=SYSM, TRACE=SYS, TRACE=SYSP, TRACE=SSCH or TRACE=SSCHP is in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'00'.
6	(6)	8	Optional timestamp; if TIMESTAMP=YES is specified, this field contains eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'5105' indicates an SSCH event.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	2	Device number.
32	(20)	4	Real address of channel program.
36	(24)	4	Virtual address of channel program.
40	(28)	4	IOSDSID.
44	(2C)	1	Condition code.
45	(2D)	12	ORB.
57	(39)	8	Dynamic seek address.
65	(31)	1	IOSGPMSK.
66	(42)	1	IOSOPT.
67	(43)	1	IOSFMSK.
68	(44)	1	Driver identifier.
69	(45)	1	IOSLEVEL.
70	(46)	1	UCBLEVEL.

SVC Comprehensive Trace Records

GTF builds SVC comprehensive trace records when an SVC interruption occurs and either TRACE=SYS or TRACe=SVC is in effect. All SVC records contain the basic data described below; however, many SVC numbers invoke additional data recording, described following the basic data.

Basic SVC Comprehensive Trace Record

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	1	Format identifier: X'01'.
6	(6)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'1000' indicates an SVC interruption.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	Jobname.
30	(1E)	8	SVC old PSW. The third and fourth bytes contain the SVC number.
38	(26)	4	Old TCB.
42	(2A)	8	CDE name.
50	(32)	4	Register 15.
54	(36)	4	Register 0.
58	(38)	4	Register 1.

GTF builds only a basic comprehensive trace record for the following SVCs:

Name	Number	Name	Number
EXIT	3	TEST	97
GETMAIN/FREEMAIN	10	SUBMIT	100
TIME	11	QTIP	101
SYNCH	12	XLATE	103
MGCR	34	TOPCTL	104
WTL	36	IMBLIB	105
TTROUTER	38	REQUEST	106
CIRB	43	MODESET	107
TTIMER	46	None	109
TTOPEN	49	DSTATUS	110
NOP	50	JECS	111
OLTEP	59	RELEASE	112
TSAV	61	SIR	113
CHATR	72	BLKPAGE	115
(IFBSTAT)	76	None	116
STATUS	79	None	117
SMFWTM	83	DSSPATCH	118
(IGC084)	84	TESTAUTH	119
SWAP	85	GETMAIN/FREEMAIN	120
EMSERV	89	None	121
VOLSTAT	91	None	122
TPUT/TGET	93	PURGEDQ	123
TSO terminal control	94	TPIO	124
SYSEVENT	95		

Basic SVC Comprehensive Trace Record with Parameter List Information: Refer to *Debugging Handbook* for details of the additional information gathered for the following SVCs:

Name	Number	Name	Number
EXCP	0	STIMER	47
WAIT/WAITR	1	DEQ	48
POST	2	SNAP	51
GETMAIN	4	RESTART	52
FREEMAIN	5	RELEX	53
LINK	6	DISABLE	54
XCTL	7	EOV	55
LOAD	8	ENQ/RESERVE	56
DELETE	9	FREEDBUF	57
ABEND	13	RELBUF/REQBUF	58
SPIE	14	STAE	60
ERREXCP	15	DETACH	62
PURGE	16	CHKPT	63
RESTORE	17	RDJFCB	64
BLDL/FIND	18	BTAMTEST	66
OPEN	19	BSP	69
CLOSE	20	GSERV	70
STOW	21	ASGNBFR/BUFINQ/RLSEBFR	71
OPEN TYPE = J	22	SPAR	73
CLOSE TYPE = T	23	DAR	74
DEVTYPE	24	DQUEUE	75
TRKBAL	25	LSPACE	78
CATLG	26	GJP	80
OBTAIN	27	SETPRT	81
CVOL	28	DISKANAL	82
SCRATCH	29	ATLAS	86
RENAME	30	DOM	87
FEOV	31	MOD88	88
ALLOC	32	TCBEXCP	92
IOHALT	33	STAX	96
WTO/WROR	35	PROTECT	98
SEGLD/SEGWT	37	Dynamic allocation	99
LABEL	39	TCAM	102
EXTRACT	40	EXCPVR	114
IDENTIFY	41		
ATTACH	42		
CHAP	44		
OVLYBRCH	45		

Minimal Trace Record

GTF builds a record that has the following format when an SVC interruption occurs and TRACE=SYSM is the GTF option in effect.

Offset		Size	Description
0	(0)	2	Length of record in bytes.
2	(2)	2	X'0000'.
4	(4)	1	Record identifier: X'FF' indicates trace record.
5	(5)	8	Optional timestamp; if TIME=YES is specified, this field contains the eight bytes of the TOD clock value.
14	(E)	2	Event identifier: X'1000' indicates an SVC interruption.
16	(10)	4	ASCB.
20	(14)	2	CPUID.
22	(16)	8	SVC old PSW. The third and fourth bytes contain the SVC number.
30	(1E)	4	Old TCB.
34	(22)	4	Register 15.
38	(26)	4	Register 0.
42	(2A)	4	Register 1.

Section 5: Diagnostic Aids

This section contains the following information to aid the reader in diagnosing GTF errors:

- A list of event identifiers (EIDs) with their symbolic names and associated events.
- A list of format identifiers (FIDs) with their associated AMDPRDMP format modules.
- A list of record identifiers (AIDs) with their associated record types.
- A description of the return codes issued by GTF modules.

Note:

Refer to *System Messages* for message number, message text, and detecting, issuing and containing modules.

Event Identifier (EID)

The event identifier (EID) in GTF trace records is a two-byte hexadecimal number that defines the event that caused the record to be built. The EID is in the form cddd where c is the event class (0-F) and ddd is the ID of the event within the class.

EID (hex)	EID (symbolic name)	Issued by	Event
0001	IEADISP1	Dispatcher	SRB dispatch.
000	IEADISP2	Dispatcher	LSR dispatch.
0003	IEADISP3	Dispatcher	TCB dispatch.
0004	IEADISP4	SVC exit prologue	SVC dispatch.
1000	IEASVCH	SVC FLIH	SVC interrupt.
2100	IECPCI	IOS	PCI interrupt.
4001	IRASRM	System Resource Manager	SYSEVENT event.
4002	IEASTAE	RTM	STAE/ESTAE event.
4003	IEAFRR	RTM	FRR event.
4004	IEAVSLSD	RTM	SLIP event.
4005	IEAVSLSU	RTM	SLIP event.
4006	IEAVSLUR	RTM	SLIP event.
5101	IECEOS	IOS	End-of-sense interrupt.
5102	IECCSCH	IOS	Clear subchannel event.
5103	IECHSCH	IOS	Halt subchannel event.
5104	IECMSCH	IOS	Modify subchannel event.
5105	IECSSCH	IOS	Start subchannel event.
5106	IECRSCH	IOS	Resume subchannel event.
5200	IECIO2	IOS	Normal I/O interrupt.
6101	IEAPINT	PFLIH	Program interrupt (codes 1-17, 19).
6200	IEATINT	PFLIH	Program interrupt (code 18).
6201	IEAEINT	EFLIH	External interrupt.
8100	ISPTPIO1	VTAM	Remote network input received as a PIU.
8200	ISPTPIO2	VTAM	Remote network output PIU.

EID (hex)	EID (symbolic name)	Issued by	Event
F100	IEASVCD	SVC Dump	SVC Dump requesting trace data.
F200	IEAAABOF	ABDUMP/SNAP	ABDUMP requesting trace data.
E000-E3FF		GTF user	User data passed by E3FF GTRACE macro.
E400-EBFF		Not Issued	Reserved.
EC00-EF9F			Reserved.
EFA0-EFA9		TCAM	
EFAA-EFAE			Reserved.
EFAF-EFE0	IMDGPD01-IMDGPD50	IBM components	
EFE1	ISTVIEID	VTAM	VTAM interval trace.
EFE2	ISTTHEID	VTAM	Reserved.
EFE3	ISTTREID	VTAM	Reserved.
EFE4	ISTTDEID	VTAM	Reserved.
EFE5-EFEE		JES2	
EFEF	ISTTPEID	VTAM	TPIOS buffer.
EFEF	ISTTPEID	VTAM	TPIOS buffer.
EFF0	ISTRPEID	VTAM	Buffer pool allocation.
EFF1	ISTCLEID	VTAM	Control layer.
EFF2	ISTLNEID	VTAM	Line trace.
EFF3	IGGSP002	SAM/PAM/DAM	
EFF4	IGGSP008	SAM/PAM/DAM	
EFF5	IDAAM01	VSAM	
EFF6	IGGSP112	SAM/PAM/DAM	
EFF7	IGGSP215	SAM/PAM/DAM	
EFF8	IGGSP119	SAM/PAM/DAM	
EFF9	IGGSP235	SAM/PAM/DAM	
EFFA	IGGSP239	SAM/PAM/DAM	
EFFB	IGGSP145	SAM/PAM/DAM	
EFFC	IGGSP251	SAM/PAM/DAM	
EFFC	IGGSP451	SAM/PAM/DAM	
EFFE	IGGSP169	SAM/PAM/DAM	
EFFF	IHLMDMA1	OPEN/CLOSE/EOV	

Format Identifier (FID)

The format identifier (FID) in GTF records is a one-byte hexadecimal number that is used to determine the name of the AMDPRDMP EDIT module that will format the record.

EID (hex)	EID (symbolic name)	Issued by	Event
00	E000-EFE4	User/component	CSECT AMDPRHEX in AMDPRREC
01-50	E000-E3FF	User	IMDUSR (01-50)
F9	EFF5	VSAM	IMDUSRF9
FD	EFEF EFF0-EFF2	VTAM	IMDUSRFD
FE	EFF3-4 EFF6-E	SAM/PAM/DAM	IMDUSRFE
FF	EFFF	OPEN/CLOSE/EOV	IMDUSRFF

System Records

FID (hex)	EID (hex)	Record Type	AMDPRDMP Format Module
00	2100	Comprehensive PCI	AMDSYS00
00	5101	Comprehensive EOS	AMDSYS00
00	5102	Comprehensive CSCH	AMDSYS00
00	5103	Comprehensive HSCH	AMDSYS00
00	5104	Comprehensive MSCH	AMDSYS00
00	5105	Comprehensive SSCH	AMDSYS00
00	5106	Comprehensive RSCH	AMDSYS00
00	5200	Comprehensive I/O	AMDSYS00
00	F100	Save hook control	AMDSYS00
00	F200	If buffer in full condition, FID=X'05'	AMDSYS05
01	1000	Comprehensive SVC	AMDSYS01
01	N/A	Timestamp	CSECT AMDPRCON in AMDPRREC
02	6101	Comprehensive PI	AMDYSS02
02	6200	Comprehensive PI	AMDSYS02
	0001		
02	0002 0003 0004	Comprehensive DSP	AMDSYS02
02	N/A	Lost data	CSECT AMDPRCON in AMDPRREC
03	1000	Minimal SVC	AMDSYS03
03	2100	Minimal PCI	AMDSYS03
03	6101	Minimal PI	AMDSYS03
03	6200	Minimal PI	AMDSYS03
03	6201	Minimal EXT	AMDSYS03
	0001		
03	0002 0003 0004	Minimal DSP	AMDSYS03
03	4001	Minimal SRM	AMDSYS03
03	4002	Minimal STAE/ESTAE	AMDSYS03
03	4003	Minimal FRR	AMDSYS03
03	N/A	Lost block	CSECT AMDPRCON in AMDPRREC
03	8100	Minimal RNIO input	AMDSYS03
03	8200	Minimal RNIO output	AMDSYS03

Record Identifier (AID)

The record identifier (AID) in GTF records is a one-byte hexadecimal number that identifies the record as a trace record or a control record.

Aid (hex)	Record Type
00	Control record.
FF	Trace record created by a GTF record build module for an event.

GTF Return Codes

Return codes are always passed in register 15.

Module	CSECT	Entry Point	Return Code	Meaning
AHLCWRIT	AHLCWRIT	AHLCWRIT	None	
AHLGTFI	AHLGTFI	AHLGTFI	None	
		AHLIESTA	None	
AHLIWRIT	AHLIWRIT	AHLIWRIT	None	
AHLMCER	AHLMCER	AHLMCER		
AHLMCIH	AHLMCIH	AHLMCIH	None	
		AHLMCIHB	None	
		AHLMCFRR	xx	Retry condition
AHLREADR	AHLREADR	AHLREADR	0	Data transferred.
			4	No data transferred; GTF inactive.
			8	No data transferred; no matching SLE.
AHLSBLOK	AHLSBLOK	AHLSBLOK	xx	ABEND conditon.
AHLSBUF	AHLSBUF	AHLSBCU1	xx	
		AHLSBUF	xx	ABEND condition.
	AHLSFEOB	AHLSFEOB	0	FEOB performed.
			4	Cannot perform FEOB; record will be lost.
AHLSCAN	AHLSCAN	AHLSCAN	0	Processing completed normally.
			8	Error in AHLSCAN.
			12	STOP command issued.
AHLSETD	AHLSETD	AHLSETD	0	All requests have been satisfied.
			4	At least 1 request satisfied; at least 1 request unsatisfied because of invalid EID or class.
			8	No requests satisfied; invalid function name or parameter list.
			12	No requests satisfied; all invalid EIDs/classes.
AHLSETEV	AHLSETEV	AHLSETEV	0	Processing completed normally.
			16	Request not satisfied due to insufficient storage.
			20	Control register 8 could not be set on one or more CPUs; MC routine can proceed.
AHLSETMG	AHLSETMG		N/A	
AHLTCCWG	AHLTCCWG	AHLTCCWG	None	
AHLTCTL1	AHLTCTL1	AHLTCTL1	0	Processing completed normally.
			12	STOP command was issued.
AHLTCTL2	AHLTCTL2		N/A	
AHLTDIR	AHLTDIR	AHLTDIR	None	
AHLTEXT	AHLTEXT	AHLTEXT	None	
		AHLTEXTE	None	
AHLTFCG	AHLTFCG	AHLTFCG	None	
AHLTFOR	AHLTFOR	AHLTFORE	None	
		AHLTFRR	None	
		AHLTRNIO	None	
		AHLTSRM	None	
		AHLTSTAE	None	

Module	CSECT	Entry Point	Return Code	Meaning
AHLTMON	AHLTMON	AHLTMON	None	
		AHLTESTA	None	
	AHLTMMSG		N/A	
AHLTPID	AHLTPID	AHLTDSP	None	
		AHLTLSR	None	
		AHLTPI	None	
		AHLTSRB	None	
AHLTPMT	AHLTPMT	AHLTPMT	0	Processing completed normally.
			12	STOP command was issued.
AHLTSCN	AHLTSCN	AHLTSCN	0	Normal completion.
			12	STOP command.
AHLTSLIP	AHLTSLIP	AHLTSLIP	None	
AHLTSVC	AHLTSVC	AHLTSVC	None	
AHLTSYFL	AHLTSYFL	AHLFIO	None	
		AHLFPI	None	
		AHLFSIO	None	
		AHLFSVC	None	
AHLTSYSM	AHLTSYSM	AHLDSP	None	
		AHLEXT	None	
		AHLPI	None	
		AHLSTAE	None	
		AHLSVC	None	
AHLTTAB	AHLTTAB	AHLTTAB	0	Normal completion.
			8	GTF termination requested.
AHLTUSR	AHLTUSR	AHLTUSR	0	Normal completion.
			8	Invalid length; less than 1 or greater than 256.
			12	Invalid data address.
			16	Invalid FID.
			20	Invalid EID.
			24	Not enough space in GTF buffers.
			28	Invalid parameter list address.
			32	Data was paged out.
AHLTXSYS	AHLTXSYS	AHLSRB	None	
		AHLSRM	None	
AHLT103	AHLT103	AHLT103	None	
AHLVCOFF	AHLVCOFF	AHLVCOFF	None	
AHLVCON	AHLVCON	AHLVCON	None	
AHLWTASK	AHLWTASK	AHLWPOST	None	
AHLWTMSG	AHLWTMSG		N/A	
AHLWTO	AHLWTO	AHLWTOMD	None	
		AHLDMPMD	'UN'	Could not take SDUMP.
			'bb'	SDUMP was issued.
AHLWWRIT	AHLWWRIT	AHLWWRIT	None	
	AHLWRMSG		None	

Section 6: GTF Macro Instructions

AHLREAD

The AHLREAD macro instruction causes scheduling of the AHLREADR routine to request transfer of trace data requested by the ABDUMP/SNAP or SVC Dump routine. It builds a parameter list for the routine (AHLREADR) which locates GTF in storage and queues the data transfer request.

AHLSTACK

The AHLSTACK macro instruction is used by the GTF record-build routines to cause the 16-byte prefix for a trace record to be constructed. The prefix is placed immediately preceding the trace data in the record-builder's work area. The record-blocking routine (AHLSBLOK) is called to initiate the blocking-buffering process.

GTRACE

IBM users and components issue the GTRACE macro instruction to cause trace data that they have collected to be placed in the GTF trace buffers. GTF will not accept user or component data unless the GTF options in effect include TRACE=USR or TRACE=USRP.

The GTRACE macro instruction has six parameters: the main storage address of the data to be recorded, the number of bytes to be recorded, the format identifier (FID) that indicates the routine that will process the user or component entry when the trace output is edited by AMDPRDMP, the ID of the user or component that built the record, specification of whether paged-out data should be obtained for recording (PAGEIN=YES/NO) and a TEST parameter. When TEST=YES is specified, GTRACE determines if GTF is active for the user event that the ID parameter specifies. The IHLMGTRC mapping macro instruction generates symbolic names equated to all assigned user/component ID values. For a description of the format of the GTRACE instruction, refer to *Service Aids*.

If PAGEIN=YES was specified, code generated by the expansion of the GTRACE macro instruction will cause the paged-out data to be paged in and the GTRACE to be issued. IF PAGEIN=NO was specified and a page fault occurs during the access of user data, GTF passes a return code of X'20' and no record is built.

The GTRACE macro instruction may be implemented in standard, list, or execute form. The standard form generates a monitor call (MC) instruction (see Section 7). The list form generates a parameter list for the GTRACE parameters, and the execute form changes the values of the parameters in the list. The TEST parameter is only valid on the standard form of the GTRACE macro.

HOOK

The HOOK macro instruction is used by system routines to communicate with GTF as follows:

- The first level interruption handlers (FLIHs), the I/O supervisor, the dispatcher, the SVC exit prolog, the system resource manager, and the recovery/termination manager issue the HOOK macro instruction to notify GTF of the occurrence of system events that GTF recognizes for tracing. A HOOK issued by these routines is called a trace hook into GTF.

  The HOOK macro instruction has two parameters used in the following format:

```
                                       P
HOOK      EID = symbolic name[,TYPE = { T  }      ]
                                       BP
                                       BT
                                       BPN
```

- EID = symbolic name — is the event identifier (EID) for the event that caused the HOOK to be issued. The EID is given as a symbolic name that is equated to a hexadecimal value. For a list of the EID symbolic names and their hexadecimal equivalents, refer to the "Diagnostic Aids" section of this chapter.

- TYPE = P — causes the HOOK macro instruction to generate a monitor call (MC) instruction (see Section 7). The assembler converts the symbolic name EID to its hexadecimal equivalent of the form X'0c0ddd', where c is the class and ddd is the displacement that is put into the MC instruction. The following MC instruction is generated when TYPE = P is the parameter:

```
Instruction 1 (MC instruction) -
  OP code:     X'AF'.
  I2:          Bits 0-3 = B'0000'.
               Bits 4-7 = c  (event class).
  B1:          B'0000'.
  D1:          ddd (event ID).
```

- TYPE = T — causes the HOOK macro instruction to generate the MC instruction shown under TYPE = P, but only if a designated assembler global switch is set in the line of the associated module assembly.

- TYPE = BP — causes the HOOK macro instruction to generate a branch into the MC routing-facility module AHLMCIH without generating an MC instruction. This parameter is used by the PFLIH for all program interruptions. The following three instructions are generated when TYPE = BP is the parameter:


```
Instruction 1 (load instruction) -
  OP code:       X'58'
  R1:            X'F' (register 15 points to AHLMCIH).
  B2:            r (base register).
  D2:            ddd (displacement).
  L 15,IEACON1

Instruction 2 (BALR instruction) -
  OP code:       X'05'
  R1:            X'E' (register 14 points to NSI after
                         execution).
  R2:            X'F' (register 15 points to AHLMCIH).

Instruction 3 (NOP instruction) -
  OP code:       X'47'.
  R1:            B'0000'.
  X2:            c (event class).
  B2:            B'0000'.
  D2:            ddd (event ID).
```

- Type=BPN — causes the HOOK macro instruction to generate a branch into the MC routing facility module AHLMCIH without generating an MC instruction. This parameter is used by non-PFLIH routines to enter module AHLMCIH through secondary entry point AHLMCIHB. The sequence of instructions generated is the same as for TYPE=BP except that the LOAD instruction is from IEACON2. (IEACON2 is a V-type address constant for AHLMCIHB.)
- Type=BT — causes the HOOK macro instruction to generate the three instructions shown under TYPE=BP, but only if a designated assembler global switch is set in the line of the associated module assembly.

IHLMGTRC

The IHLMGTRC macro instruction is mapping macro that generates symbolic names equated to hexadecimal values for IBM components that use the GTRACE macro instruction. The component specifies its assigned symbolic name as the operand of the ID=keyword in the GTRACE macro instruction.

SETEVENT

The SETEVENT macro instruction is issued by users of the MC instruction (for example, GTF) as an interface to the MC routine facility. The MC routine facility passes control to the proper event handler when a requested MC instruction is issued. Expansion of the macro instruction causes building of a parameter list for the SETEVENT service. This service then does the actual manipulation of the MC routing facility to perform the requested function.

Section 7: Monitor Call (MC) Instruction

The monitor call (MC) instruction is a maskable interruption instruction. Both the HOOK and GTRACE macro instructions generate the MC instruction to pass control to GTF. The format of the MC is:

SI format – MC $D_1(B_1),I_2$

AF	I_2	B_1	D_1	
0	8	16	20	31

Each monitoring event is defined by the low order four bits of the I2 filed, known as the class of the event, and by the sum of D_1 and B_1. The GTF event identifier (EID) is two bytes of the form cddd where c is the class of the event and ddd is the event within the class. The GTF EID corresponds to the fields of the MC instruction as follows:

MC instruction format

AF	I_2	B_1	D_1	
0	8	16	20	31

MC instruction for GTF

AF	0	c	0	ddd	
0	8	12	16	20	31

(Monitor call instructions generated by either the HOOK or GTRACE macro instructions for GTF always contain a B_1 value of zero.)

Chapter 2. Module Map and IDR List (AMBLIST)

Section 1: Introduction

AMBLIST is a linkage-editor service aid program that provides the user with formatted listings for use in problem determination. AMBLIST operates as a problem program under MVS/XA.

Functions

The main functions of AMBLIST are:

- Producing formatted object module listings.
- Producing formatted load module listings.
- Producing formatted nucleus listings.
- Producing load module maps and cross-reference listings.
- Formatting the information in the CSECT identification records (IDRs) of a load module.
- Producing a formatted link pack area map.

An object module listing contains the text of the module, that is, its instructions and data in hexadecimal representation, and the END record. It also contains the external symbol dictionary (ESD) and the relocatable load dictionary (RLD).

The contents of a load module listing vary according to the options selected by the user. A load module listing (including the nucleus) may contain the control and test records of the stored load module, including the external symbol dictionary and the relocatable load dictionary, scatter and translation tables, IDRs, a module map, and a cross-reference listing.

The CSECT identification records (IDRs) can be formatted and listed. The user can request that all IDRs be displayed or that just those IDRs containing AMASPZAP service aid program data and those containing optional user-supplied data be displayed. If a translator supports IDR, the translator, its version and modification level, and the date of translation for a particular control section are recorded in the IDR translator data records (subtype 04). The IDR linkage-editor record (subtype 02) contains information pertaining to the linkage-editor that constructed the load module: its version, modification level, and date of load module creation. AMBLIST also lists any user-supplied data associated with the executable code of a control section recorded in IDR user data records (subtype 08). The modifications that AMASPZAP performed on a specified control section of a load module, including the date and the specific data modified, are recorded in the IDR AMASPZAP data records (subtype 01).

Environment

AMBLIST resides in the system library. AMBLIST is executed as a job step through specification on an EXEC job control statement in the input stream; or it receives control through the execution of a LINK, LOAD, ATTACH, or XCTL macro instruction.

Input to AMBLIST consists of object modules or load modules plus control statements. Control statement input is defined by the SYSIN DD statement. The DDN option in a control statement (or the default name of SYSLIB) specifies the name of the DD statement defining the sequential or partitioned data set containing the module(s) optionally designated as input in the control statement. Output from AMBLIST is the listings requested by the control statement plus any diagnostic messages. The output from AMBLIST goes to any user-defined data set defined in the SYSPRINT DD statement.

Storage Requirements

AMBLIST is operational in the minimum configuration required for MVS/XA. It requires a minimum of 64K bytes of storage. Static data and instructions require 36K bytes of storage, the remainder of the 64K bytes of storage is dynamic storage, including an allowance for non-resident access method routines.

Physical Characteristics

AMBLIST is a reenterable program and consists of a control module, five processing modules, two diagnostic modules, and an open exit module for SYSPRINT:

- Control module: HMBLKCTL.
- Load module processing module: HMBLKLDM.
- Map and cross-reference processing module: HMBLKXRF.
- Object module processing module: HMBLKOBJ.
- Link pack area mapping module: HMBLKLPA.
- CSECT identification record processing module: HMBLKIDR.
- Error message writer: HMBLKERR.
- Text of error messages: HMBLKMSG.
- Open exit module for SYSPRINT: HMBLKSZE.

Each of these modules consist of a single external procedure, except for HMBLKMSG, which is a CSECT containing two tables.

HMBLKCTL examines the input stream control statement (from the SYSIN data set) and determines which of the five processing modules must be called.

Operational Considerations

Operation of AMBLIST depends upon the type of input received and on the user options specified. The control statements in the input stream re in the format of an operation and its applicable operands. The formats are shown in *Service Aids*. The control routine scans the control statement and turns on the appropriate switches in the OPTNMAP map to indicate which routines are to be used.

Section 2: Method of Operation

This section shows how AMBLIST provides the user with formatted listings. As shown in LIST Figure 2-1, the description begins with an overview, then describes the following stages of overall processing:

- Control processing (MO1).
- Listing processing (MO2 - MO5).
- Diagnostic processing (MO6).

Reading Method of Operation Diagrams

Method of operation diagrams are arranged in an input-processing-output layout; the left side of the diagram contains the data that serves as input to the processing steps in the center of the diagram, and the right side contains the data that is output from the processing steps. Each processing step is numbered; the number corresponds to the verbal description of the step in the extended description. While the processing step in the diagram is in general terms, the corresponding text is a specific description that includes a cross-reference to the code for the processing.

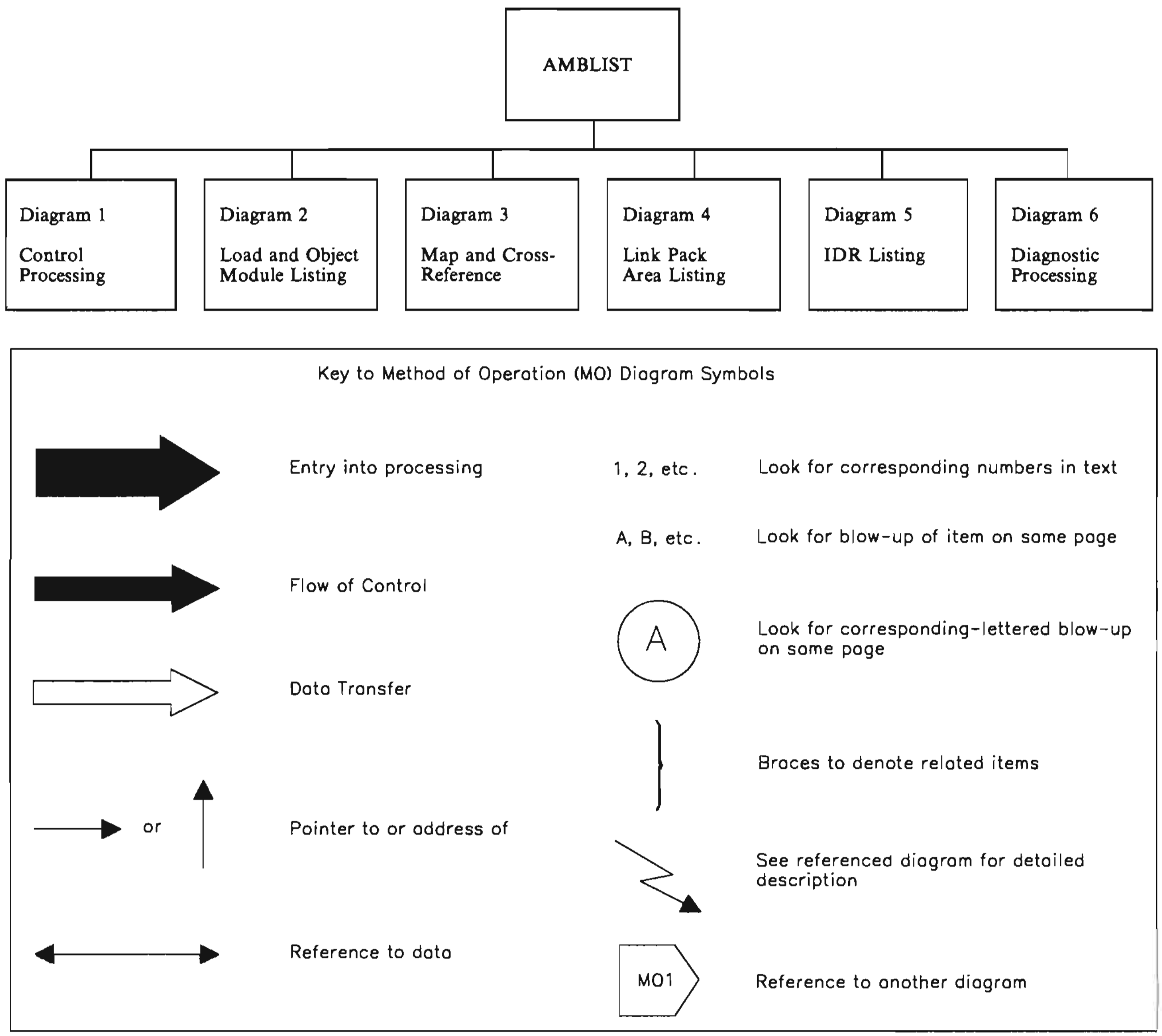
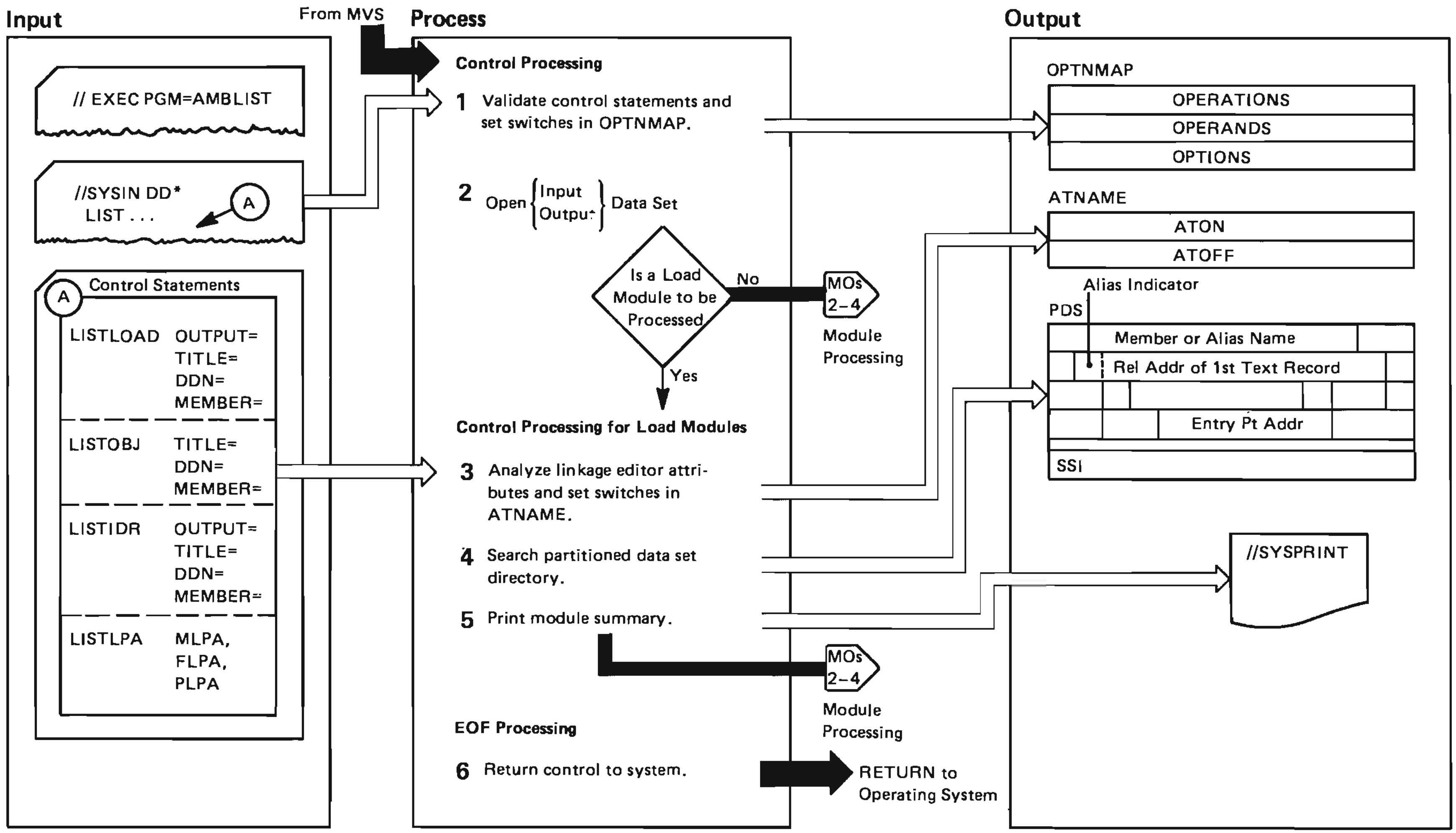


Figure 2-1. Key to Method of Operation Diagrams for ABMLIST

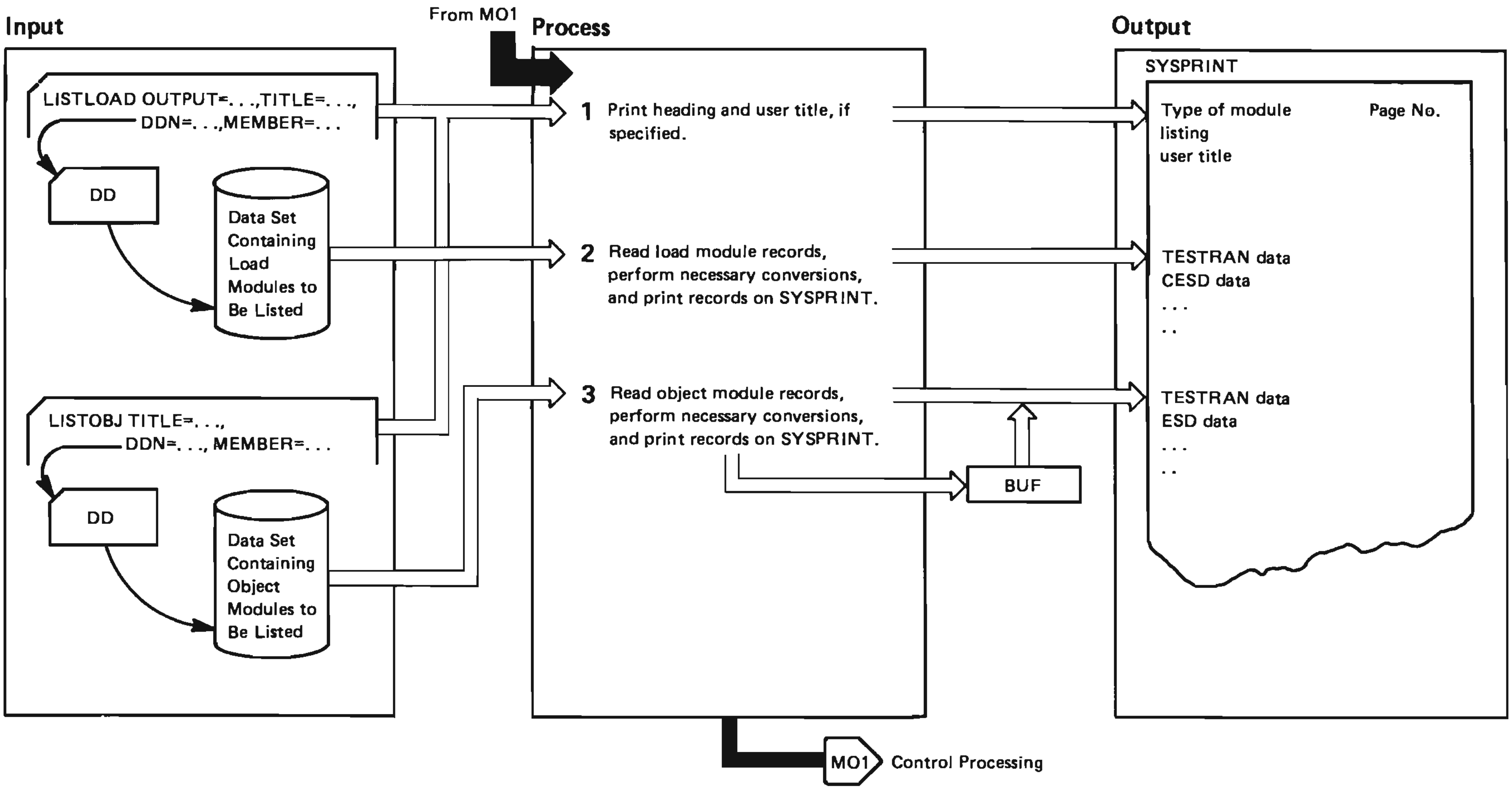
LIST Diagram 1. AMBLIST Control Processing (Part 1 of 2)



LIST Diagram 1. AMBLIST Control Processing (Part 2 of 2)

Extended Description	Module	Label
1 The user can specify four operations in the AMBLIST control statement. Each of these operations has several associated operands. See (A) for a diagram of the operations and operands. (A detailed discussion of the operations, operands, and options associated with the operands and their meanings is given in *Service Aids.*) HMBLKCTL analyzes and validates the control statements and sets the appropriate switches in OPTNMAP.	HMBLKCTL	AAH1
• If LISTLOAD or LISTIDR is specified, HMBLKCTL performs the processing listed in steps 2, 3, and 4.	HMBLKCTL	AAH1
• When LISTOBJ is specified on the control statement, HMBLKCTL performs step 2.	HMBLKCTL	AAH1
• If an error is discovered, HMBLKCTL passes control to the error processing routine HMBLKERR (MO 5).		
2 There is one output data set SYSPRINT, which contains all the output of AMBLIST.	HMBLKCTL	AAA1
3 The linkage editor attributes specified in the directory entry are analyzed and the appropriate switches are set in ATNAME.	HMBLKCTL	
4 HMBLKCTL scans the partitioned data set (PDS) directory for aliases and secondary entry point addresses, for the system status index (SSI), for the access code (APF), and for the main entry point address of the load module.	HMBLKCTL	PDSREAD
5 A module summary is prepared by HMBLKCTL for the first page of an output listing for a load module. (A detailed discussion of the contents of the module summary is found in *Service Aids.*)	HMBLKCTL	CHOW
6 When there are no more control statements to be processed, dynamic storage is released and control is returned to the operating system.	HMBLKCTL	ENDIT

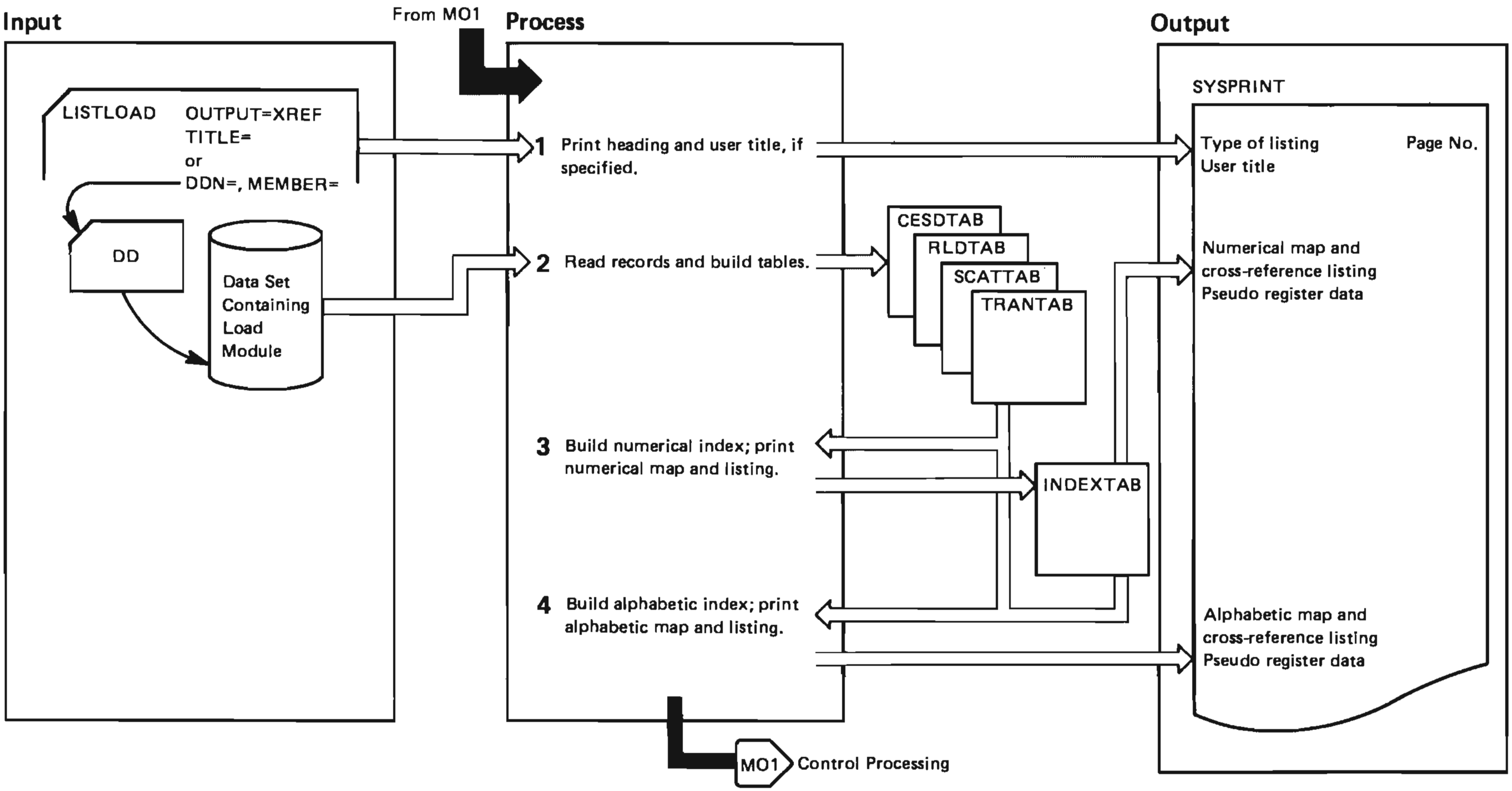
LIST Diagram 2. Load and Object Module Listing (Part 1 of 2)



LIST Diagram 2. Load and Object Module Listing (Part 2 of 2)

Extended Description	Module	Label
1 The address of the data set containing the module(s) to be listed and the address of SYSPRINT are obtained from the parameter list passed from HMBLKCTL. The heading, indicating which type of module listing follows, and the user title, if specified, are written in SYSPRINT.	HMBLKLDM/ HMBLKOBJ	BEGIN/ INITID
2 The records are read in order from the data set specified on the control statement. Some data items within the records are converted. See the Data Conversions table for details. Finally, the records are written in SYSPRINT. (Records are read, converted, and printed one by one.)	HMBLKLDM/ HMBLKOBJ	
3 After all records have been listed, control is returned to HMBLKCTL. If error conditions arose during processing they are indicated in the ERRORS bit map in HMBLKCTL, and diagnostic processing ensues. Otherwise, HMBLKCTL handles the next user request.	HMBLKLDM/ HMBLKOBJ	EOJX/ OBJEOF

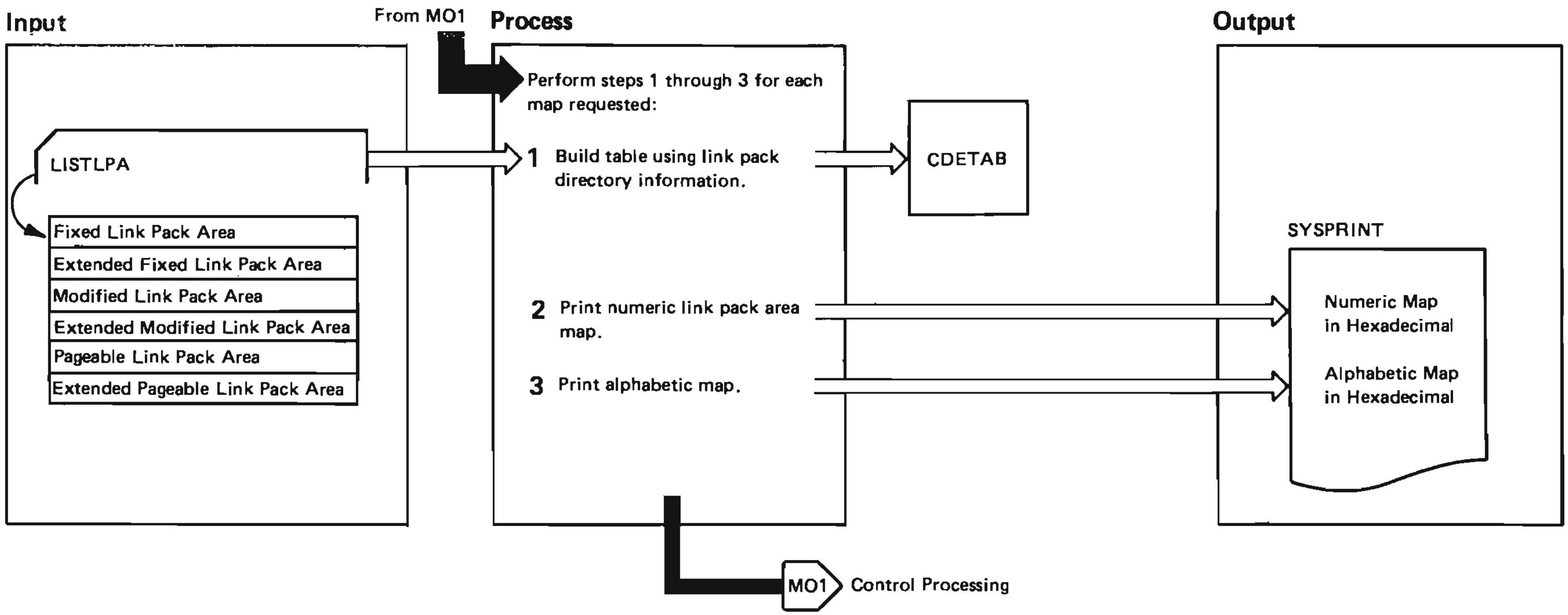
LIST Diagram 3. Map and Cross-Reference Listing (Part 1 of 2)



LIST Diagram 3. Map and Cross-Reference Listing (Part 2 of 2)

Extended Description	Module	Label
1 The address of the data set containing the load module to be processed and the address of SYSPRINT are obtained from the parameter list passed from HMBLKCTL.	HMBLKXRF	AUTOINIT
The heading, indicating which type of listing follows and the user title, if specified, are printed on SYSPRINT.		WRITE1
2 Read load module, enter CESD records in CESDTAB, enter RLD records in RLDTAB. If the load module is a nucleus, enter translation records into TRANTAB, enter scatter records into SCATTAB. All other records are ignored. If any table overflows, processing is terminated and control is returned to HMBLKCTL.	HMBLKXRF	READ
When the load module is a nucleus, CESDTAB and RLDTAB addresses are converted to loaded addresses using the SCATTAB and TRANTAB entries. End of input is indicated by the end-of-module (EOM) indication.	HMBLKXRF	NUC
3 The CESDTAB is sorted into ascending numerical sequence according to the segment number and the address of the CESDTAB symbol. The INDEXTAB is built using the CESDID of each CESDTAB entry.	HMBLKXRF	NCSORT
The RLDTAB is sorted into ascending numerical sequence according to the address contained in the RLDTAB entry.		NRSORT
The segment numbers (in CESDTAB entries) are converted to the EBCDIC code for decimal digits; addresses (in CESDTAB or RLDTAB entries) are converted to EBCDIC code for hexadecimal digits.	HMBLKXRF	NACESD
A numeric module map and cross-reference are then printed. If the load module is in overlay format, the above data is printed segment by segment, beginning with segment 1. Finally, pseudo register data is printed.		WRITE
4 The CESDTAB is resorted into ascending alphabetic sequence according to the CESDTAB symbol. The INDEXTAB is rebuilt using the CESD ID of each CESDTAB entry.	HMBLKXRF	ACSORT
Data is converted as in step 3.	HMBLKXRF	NACESD ARLD
An alphabetic module map is then printed. To print an alphabetic cross-reference listing ordered by symbol referred to, R pointers (from the RLDTAB) are sought to match ESD identifiers in the CESDTAB entries. When a match is found, a cross-reference line is converted and printed.		WRITE
After all listings have been printed, control is returned to HMBLKCTL. If error conditions arose during processing, they are indicated in the ERRORS bit map, and diagnostic processing ensues. Otherwise, HMBLKCTL handles the next user request.	HMBLKXRF	FINISH

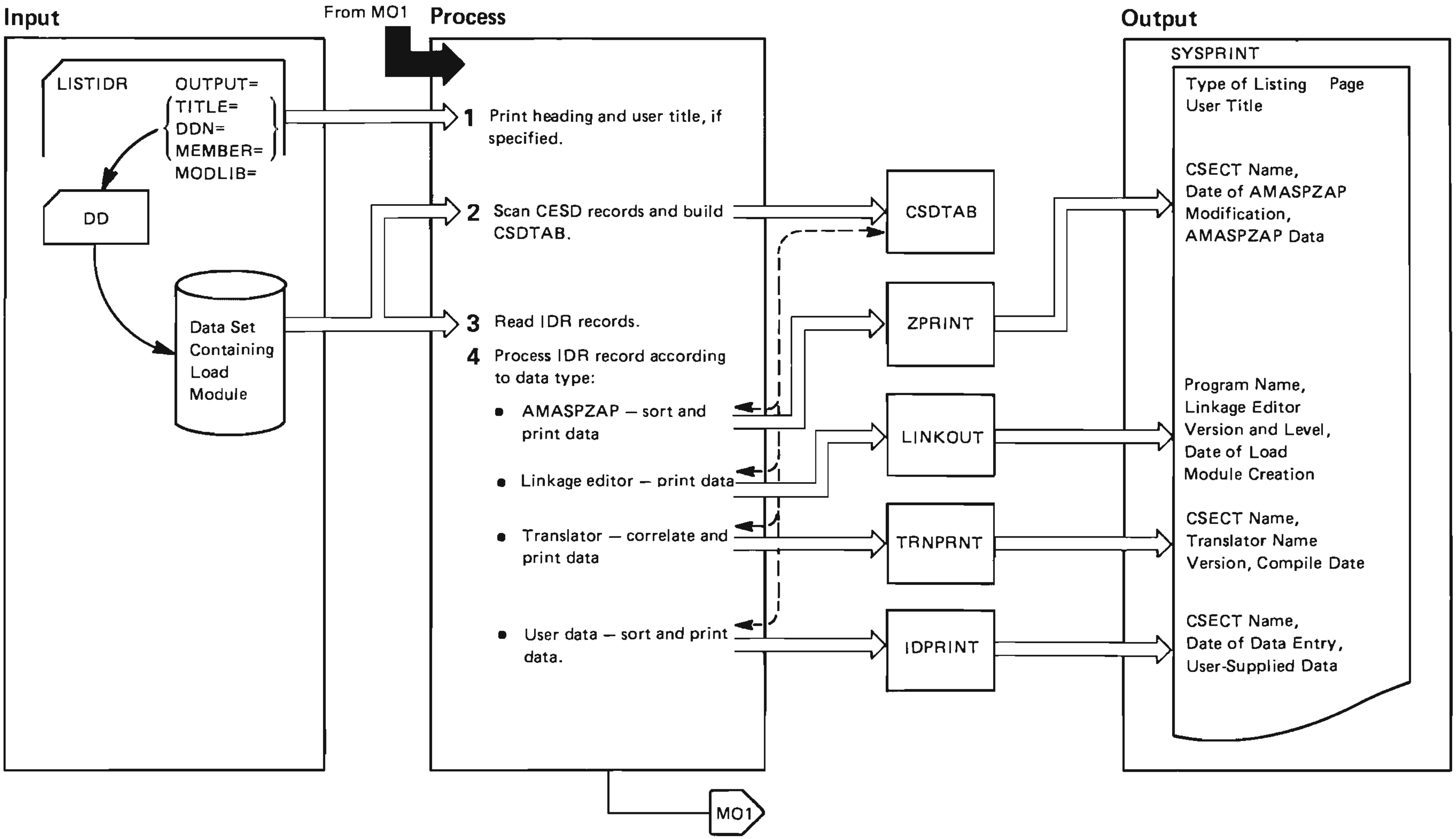
LIST Diagram 4. Link Pack Area Map Listing (Part 1 of 2)



LIST Diagram 4. Link Pack Area Map Listing (Part 2 of 2)

Extended Description	Module	Label
1 If the FLPA and/or MLPA is being processed, HMBLKLPA searches the active link pack directory entry chain for entries pointing to modules in the FLPA and/or MLPA and their extensions. Otherwise, HMBLKLPA uses the pointer in the CVT (CVTLPDIR) to get the entries for the PLPA and its extension.	HMBLKLPA	HMBLKLPA
AMBLIST builds the CDETAB using the link pack directory information. The name, entry point, and length are moved to the CDETAB.	HMBLKLPA	
2 The CDETAB is sorted numerically by module location.		
3 The CDETAB is re-sorted alphabetically by module name.		

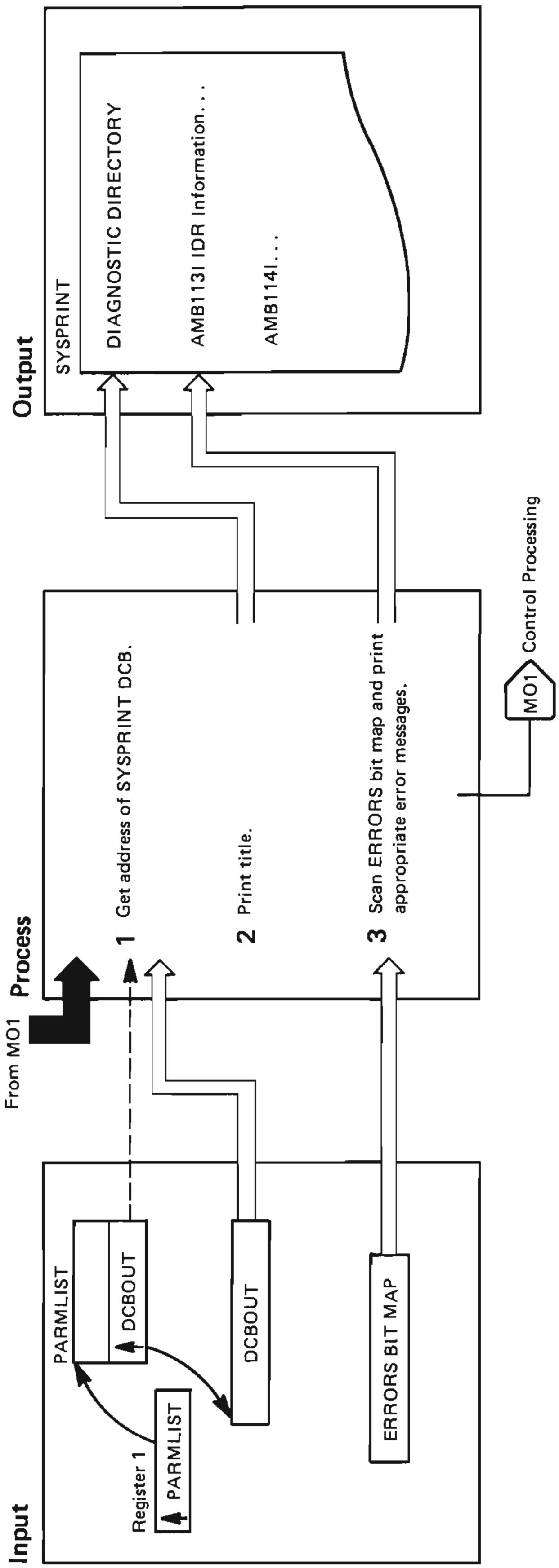
LIST Diagram 5. IDR Listing (Part 1 of 2)



LIST Diagram 5. IDR Listing (Part 2 of 2)

Extended Description	Module	Label
1 The address of the data set containing the load module whose IDR's are to be listed and the address of SYSPRINT are obtained from the parameter list passed from HMBLKCTL. If the MODLIB keyword was specified, only user data and AMASPZAP data are printed. Printing of the module summary is suppressed. The head, indicating which type of listing follows and the user title, if specified, are printed on SYSPRINT.	HMBLKIDR	PRNT1
2 Only CESD records containing SD (section definition) or PC (private code) entries are processed. The symbolic names and ESD identifiers from these records are entered in the CSDTAB table.	HMBLKIDR	CESDRT
3 Read IDR records and determine what type of data they contain and then process accordingly.	HMBLKIDR	IDRTN
4 In the following processing, the ESD identifier from the IDR is used to search the CSDTAB. The CSECT name associated with the matching ESD identifier in the CSDTAB is then printed with the IDR data.		
• AMASPZAP data – Get CSECT name from CSDTAB and enter it and AMASPZAP data in SORTAB, sort SORTAB alphabetically by CSECT name, and print data from the ZPRINT output area to SYSPRINT.	HMBLKIDR	ZAPRT
• Linkage editor data – The data is printed from the LINKOUT output area to SYSPRINT.	HMBLKIDR	LKERT
• Translator data – Get CSECT name from CSDTAB and enter it in TRNTAB. Enter translator data in TDTAB; correlate TRNTAB and TDTAB entries and print data from the TRNPRNT output area to SYSPRINT.	HMBLKIDR	TR1
• User data – Get CSECT name from CSDTAB and enter it and user data in IDENTDAT, sort IDENTDAT alphabetically by CSECT name, and print data from the IDPRINT output area.	HMBLKIDR	IDENRT
The last IDR is identified by a flag in the subtype field. After all IDRs have been listed, control is returned to HMBLKCTL. If error conditions arose during processing, they are indicated in the ERRORS bit map in HMBLKCTL, and diagnostic processing ensues. Otherwise, HMBLKCTL handles the next user request.	HMBLKIDR	RET

LIST Diagram 6. Diagnostic Processing (Part 1 of 2)



LIST Diagram 6. Diagnostic Processing (Part 2 of 2)

Extended Description	Module	Label
1 HMBLKERR obtains the address of the SYSPRINT DCB from the parameter list (PARMLIST) passed from HMBLKCTL.	HMBLKERR	
2 HMBLKERR then prints the title 'Diagnostic Directory' at the top of a listing page.	HMBLKERR	
3 The bits in ERRORS, a bit map defined in HMBLKCTL are checked. Bits were set to 1 if HMBLKCTL, HMBLKLDM, HMBLKXRF, HMBLKOBJ, or HMBLKIDR detected errors.	HMBLKERR	
If a bit is set to 1, the associated message text is obtained from the MESSAGES and MSGLIST tables in HMBLKMSG and printed on SYSPRINT. After all error messages have been printed, control is returned to HMBLKCTL and the next user request is processed.	HMBLKERR	

Section 3: Program Organization

This section describes the organization of the AMBLIST program. It contains:

- A description of the module organization (see Figure 2-2).
- Descriptions of each AMBLIST module, in alphabetical order by object module name.

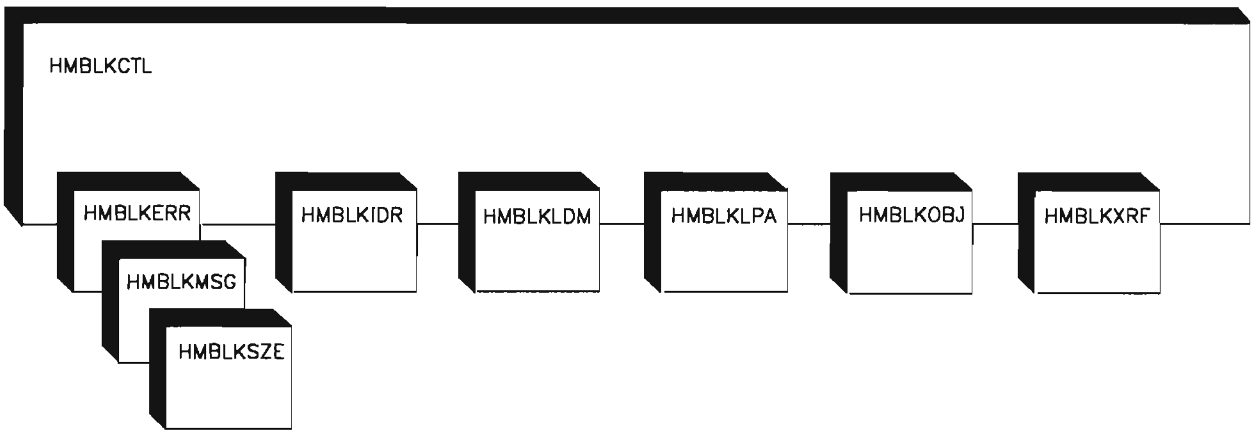


Figure 2-2. AMBLIST Module Organization

Module Descriptions

HMBLKCTL — Control Module

Entry from: The scheduler when AMBLIST is invoked.

Data areas defined or updated: OPTNMAP, ERRORS.

Routines called: HMBLKLDM, HMBLKPA, HMBLKXRF, HMBLKOBJ, HMBLKIDR, HMBLKERR.

Exits: At end-of-file for the SYSIN data set, return to the operating system.

MOs: 1.

Operation: Validates the control statement and passes control to the appropriate processing routine. It also passes control to the error message printer (HMBLKERR).

HMBLKERR — Error Message Printer

Entry from: HMBLKCTL.

Data areas defined or updated: None.

Routines called: MESSAGES and MSGLIST in HMBLKMSG.

Exits: Return to HMBLKCTL.

MOs: 6.

Operation: Obtains the message text from HMBLKMSG and prints the message in SYSPRINT.

HMBLKIDR — CSECT Identification Record Processing Module

Entry from: HMBLKCTL.

Data areas defined or updated: CSDTAB, IDENTAB, SORTAB, TDBAB, TRNTAB.

Routines called: None.

Exits: Return to HMBLKCTL.

MOs: 5.

Operation: Reads IDR records, correlates CSECT names with data if necessary, and prints the IDR data in SYSPRINT.

HMBLKLDM — Load Module Processing Module

Entry from: HMBLKCTL.

Data areas defined or updated: None.

Routines called: None.

Exits: Return to HMBLKCTL.

MOs: 2.

Operation: Reads load module records, performs conversions, and prints the records in SYSPRINT.

HMBLKLPA — Link Pack Area Mapping Module

Entry from: HMBLKCTL.

Data areas defined or updated: CDETAB.

Routines called: None.

Exits: Return to HMBLKCTL.

MOs: 4.

Operation: Products a formatted listing of the link pack directory entries that reside in the fixed link pack area, modified link pack area, pageable link pack area, and their extensions.

HMBLKMSG — Text of Error Messages

Entry from: HMBLKERR.

Entry point names: MESSAGES, MSGLIST.

Data areas defined or updated: MESSAGES, MSGLIST.

Routines called: N/A.

Exits: N/A.

MOs: 5.

Operation: This nonexecutable module contains the two tables MESSAGES and MGLIST.

HMBLKOBJ — Object Module Processing Module

Entry from: HMBLKCTL.

Data areas defined or updated: None.

Routines called: None.

Exits: Return to HMBLKCTL.

MOs: 2.

Operation: Reads object module records, performs conversions, and prints the records in SYSPRINT.

HMBLKSZE — Open Exit Module for SYSPRINT

Entry from: HMBLKCTL.

Data areas defined or updated: None.

Routines called: None.

Exits: Return to HMBLKCTL.

MOs: None.

Operation: This module is an exit list routine for the SYSPRINT DCB.

HMBLKXRF — Map and Cross-Reference Processing Module

Entry from: HMBLKCTL.

Data areas defined or updated: CESDTAB, RLDTAB, INDEXTAB, TRANTAB, SCATTAB.

Routines called: None.

Exits: Return to HMBLKCTL.

MOs: 3.

Operation: Reads load module records, sorts, CESD and RLD records, performs conversions and lists the CESD and RLD records both alphabetically and numerically by address. Produces map and cross-reference listing of the nucleus.

Section 4: Data Areas

This section contains descriptions of the following data areas built and used by AMBLIST modules:

- CDETAB
- CESDTAB
- CSDTAB
- ERRORS
- IDENTDAT
- INDEXTAB
- MESSAGES
- MSGLIST
- OPTNMAP
- RLDTAB
- SCATTAB
- SORTAB
- TDTAB
- TRANTAB
- TRNTAB

CDETAB

Size: 21 bytes.

Created by: HMBLKLPA.

Updated by: HMBLKLPA.

Offset		Size	Description		
0	(0)	8	Module name.		
8	(8)	8	Major name or length and location.		
8	(8)	4	Length of module.		
12	(C)	4	Location of module.		
16	(10)	4	Entry point address.		
20	(14)	1	**Bit settings**		**Meaning**
			1...	....	Minor entry.
			.1..	....	Major entry.
			..xx	xxxx	Reserved.

CESDTAB

Size: 19 bytes.

Created by: HMBLKXRF.

Updated by: HMBLKXRF.

Offset		Size	Description
0	(0)	2	ESD identifier of symbol from CESD.
2	(2)	8	8-character symbolic name.

Offset		Size	Description	
10	(A)	1	**Bit settings**	**Meaning**
			xxxx 0000	Section definition (SD).
			xxxx 0011	Label reference (LR).
			xxxx 0100	Private code (PC).
			xxxx 0101	Common (CM).
			xxxx 0110	Pseudo register (PR).
			xxxx 0010	External reference (ER).
			xxxx 1010	Weak external reference (WX).
			xxx1 x100	Private code marked delete (PD).
11	(B)	4)	Address of symbol.	
15	(F)	1	Segment number in which the symbol appears.	
16	(10)	3	Either chain ID or length of the CSECT from CESD.	

CSDTAB

Size: 12 bytes.

Created by: HMBLKIDR.

Updated by: HMBLKIDR.

Offset		Size	Description
0	(0)	8	CSECT name from CESD.
8	(8)	2	Reserved.
10	(A)	2	ESD identifier from CESD.

Errors

Size: 32 bits.

Created by: HMBLCKCTL.

Updated by: HMBLKLDM, HMBLKLPA, HMBLKXRF, HMBLKOBJ, HMBLKIDR, HMBLKCTL.

Size: HMBLKERR.

Errors is a 32-bit map. Bits 0 through 31 are correlated to error messages 1 through 32.

IDENTDAT

Size: 51 bytes.

Created by: HMBLKIDR.

Updated by: HMBLKIDR.

Offset		Size	Description
0	(0)	8	CSECT name associated with the ESD identifier in the IDR record.
8	(8)	3	Date on which user data was supplied to the load module via the linkage editor identify function.
11	(B)	40	User-supplied data as specified in the linkage editor identify control statement.

INDEXTAB

Size: 2 bytes.

Created by: HMBLKXRF.

Updated by: HMBLKXRF.

Offset		Size	Description
0	(0)	2	Pointer to a CESD table line. An ID chain pointer, R pointer, or P pointer is used as an offset into the index table to find a value which is used as an offset into the CESD table.

MESSAGES

Defined in: HMBLKMSG.

Used by: HMBLKERR.

MESSAGES is a table containing the text of the error messages. The length of an entry (MSGxx) in MESSAGES can be obtained from the second halfword of the corresponding entry in MSGLIST. For example, the length of MSG01 can be obtained from LST01 in MSGLIST.

MSGLIST

Size: Maximum of 32 4-byte entries.

Defined in: HMBLKMSG.

Used by: HMBLKERR.

Offset		Size	Field	Description
0	(0)	4	LST01	The first two bytes contain the offset in bytes of the message text from the beginning of MESSAGES. The second 2 bytes contain the length of the message text in MESSAGES.
4	(4)	4	LST02	Same as above.

OPTNMAP

Size: 32 bits

Created by: HMBLKCTL.

Updated by: HMBLKCTL.

Used by: HMBLKCTL.

Bit	Field	Description
0	LISTLOAD	LISTLOAD operation specified.
1	LISTOBJ	LISTOBJ operation specified.
2	LISTIDR	LISTIDR operation specified.
3	LISTLPA	LISTLPA operation specified.

Bit	Field	Description
4	TITLE	TITLE operand specified.
5	DDN	DDN operand specified.
6	MEMBER	MEMBER operand specified.
7	MODLIST	LIST option specified.
8	XREF	XREF option specified.
9	IDENT	IDENT option specified.
10	PO	Partitioned data set (PDS).
11	LASTMEM	Last member name indicator.
12	LIMIT	Too may member names.
13	MCONTIN	MEMBER option continuation.
14	RELOC	Relocation factor given.
15-31	Reserved.	

RLDTAB

Size: 9 bytes.

Updated by: HMBLKXRF.

Used by: HMBLKXRF.

Offset		Size	Description
0	(0)	2	Entry number of the CESD entry (or translation entry) that indicates which symbol value is to be used in the computation of the address constant's value.
2	(2)	2	Entry number of the CESD entry (or translation table entry) that indicates which CSECT contains the address constant.
4	(4)	1	Flags: When byte format is xxxx LLST, xxxx specifies the type of this RLD item (address constant): 0000 — nonbranch-type is assembler language. DC A (name). 0001 — branch-type in assembler language. DC V (name). 0011 — pseudo register cumulative displacement value. 1000 and 1001 — this address constant is not be replaced because it refers to an unresolved symbol. LL specifies the length of the address constant: 01 — two-byte. 10 — three-byte. 11 — four-byte. S specifies the direction of relocation: 0 — positive 1 — negative. T specifies the type of the next RLD item: 0 — the following RLD item has a different relocation and/or position pointer. 1 — the following RLD item has the same relocation and position pointers as this and therefore is omitted.
5	(5)	4	Address of the address constant.

SCATTAB

Size: 4 bytes.

Created by: HMBLKXRF.

Offset		Size	Description
0	(0)	1	RSECT, RMODE flag byte .... 1... RSECT. .... .1.. RMODE. xxxx ..xx Reserved.
1	(1)	4	Address assigned by linkage editor to a control section (SD, PC, or CM).

SORTAB

Size: 19 bytes.

Created by: HMBLKIDR.

Used by: HMBLKIDR.

Offset		Size	Description
0	(0)	8	CSECT name associated with the ESD identifier in the IDR record.
8	(8)	3	Date on which AMASPZAP modification was performed on module.
11	(B)	8	Data specified during AMASPZAP processing.

TDTAB

Size: 15 bytes.

Created by: HMBLKIDR.

Used by: HMBLKIDR.

Offset		Size	Description
0	(0)	10	Translation name.
10	(A)	2	Version and modification level information (VVMM).
12	(C)	3	Date of translation (YYDDD).

TRANTAB

Size: 2 bytes.

Created by: HMBLKXRF.

Offset		Size	Description
0	(0)	2	Pointer to the scatter table entry that contains the address of the control section containing this CESD table entry.

TRNTAB

Size: Variable number of 8-byte entries.

Created by: HMBLKIDR.

Used by: HMBLKIDR.

Offset		Size	Description
0	(0)	8	CSECT name associated with the ESD identifier in the IDR record.
8	(8)	8	Same as above.

The end of the table is marked by blanks.

Section 5: Diagnostic Aids

This section contains the following information that will help to diagnose problems in AMBLIST:

- A summary of register activity.
- A description of the SYNAD routines of the AMBLIST control module.

Note: Refer to *System Messages* for message numbers, message text, and detecting, issuing, and containing modules.

Register Activity

HMBLKCTL

Register	Use
1	Pointer to parameter lists.
2	Third base register for addressing static data and code.
4	Pointer to BLDL list.
5	Base register for addressing DCB DSECT.
6	Base register for addressing PDS directory DSECT.
7	Second base register for addressing static data and code.
9	Column pointer for scanning control cards.
11	First base register for addressing static data and code.
12	Base register for addressing the dynamic storage area.
13	Save area address.
14	Linkage register for internal calls.
15	Branch register for external calls.

HMBLKIDR

Register	Use
1	Pointer to parameter list and to output DCB.
2	Pointer to input buffer and to data to be converted.
3	Pointer to input DCB.
11	Base register for addressing static data and code.
12	Base register for addressing the dynamic storage area.
13	Save area address.
15	Branch register for external calls.

HMBLKLDM

Register	Use
1	Pointer to parameter list and to output DCB.
2	Pointer to input DCB or a work register.
3	Pointer to input buffer or a work register.
4	Return address from print subroutine.
5	Return address from conversion subroutine.
6	Line count.
7	Loop control register.
11	Base register for addressing static data and code.
12	Base register for addressing the dynamic storage area.
13	Save area address.
14	Linkage register for internal calls.
15	Branch register for external calls.

HMBLKLPA

Register	Use
1	Pointer to parameter list.
2	Base for contents directory block.
10	Pointer to packed data.

HMBLKOBJ

Register	Use
1	Pointer to parameter list.
5	Pointer to input buffer and to output DCB, or work register.
6	Pointer to output buffer and to input DCB, or work register.
9	First base register for addressing static data and code.
11	Second base register for addressing the static data and code.
12	Base register for addressing the dynamic storage area.
13	Save area address.
14	Linkage for calls.
15	Branch register for external calls.

HMBLKXRF

Register	Use
1	Pointer to parameter list and to output buffer.
2	CVD instruction register, loop control register, pointer to input DCB and to output DCB, or index for sort.
3	Pointer to input buffer or return address for branches within HMBLKXRF.
4	Third base register for addressing static data and code.
5	Pointer to current CESD table line being processed.
9	Second base register for addressing the static data and code.
11	First base register for addressing the static data and code.
12	Base register for addressing the dynamic storage area.
13	Save area address.
14	Linkage register for internal calls.
15	Used as branch register for external calls.

SYNAD Routine — HMBLKCTL

When HMBLKCTL detects an uncorrectable input or output error, control is passed to an error analysis (SYNAD) routine. If the error occurs while processing a partitioned data set, the SYSIOPDS routine receives control and issues a WTO macro instruction to display an error message at the console. Control is then returned to the system.

Chapter 3. Print Dump (AMDPRDMP)

Section 1: Introduction

The AMDPRDMP service aid program formats and prints the contents of the AMDSADMP output data set, the SYS1.DUMPnn ABEND dump data set, or any dumps data sets produced by SVC dump, and the GTF trace data set.

The user controls the operation of AMDPRDMP with JCL statements and AMDPRDMP control statements.

Job Control Language Statements

JCL statements allow the user to describe:

- The input data sets, SYSIN and SYSTSIN
- The AMDPRDMP table of contents data set, INDEX
- The AMDPRDMP output data set, PRINTER
- The AMDPRDMP message data sets, SYSPRINT and SYSTPRT
- The direct access work data set, SYSUT1
- A data set for transfer of a dump data set, SYSUT2
- The data set that contains the dump, TAPE and/or a user specified DDname

The PARM parameter on the EXEC statement allows the user to specify:

- A title for the AMDPRDMP output listing to be requested from the console
- The number of lines per page to be printed in the output listing
- The option of stopping the AMDPRDMP processing for a function
- The action to be taken if an error occurs in an exit or format routine that is editing GTF records

For a detailed description of the JCL statements for AMDPRDMP, see *MVS/XA SPL: Service Aids.*

AMDPRDMP Control Statements

Control statements allow the user to specify formatting. The user enters the control statements into the system through either the SYSIN data set or the console.

AMDPRDMP control statements introduce optional functions into this AMDPRDMP operation. The following are the AMDPRDMP control statements:

- ASMDATA — causes ADMPRDMP to format and print the contents of auxiliary storage manager (ASM) control blocks that exist in the input data set.
- CPUDATA — causes AMDPRDMP to format and print data related to each processor in a dumped system.
- CVT — provides the address of the communications vector table if there is reason to believe that the CVT pointer at virtual storage location X'4C' in a virtual storage dump has been destroyed.
- CVTMAP — causes AMPRDMP to format and print the communications vector table (CVT).
- DAEDATA — causes AMDPRDMP to format and print the dump symptoms that DAE analyzes in determining whether or not to suppress the dump.
- EDIT — causes AMDPRDMP to obtain, format, and print trace data created by the generalized trace facility (GTF).
- END — terminates AMDPRDMP processing.
- FORMAT — causes AMDPRDMP to format and print the contents of the major system control blocks associated with each address space in a dumped system.

  The control statement provides for compatibility with prior versions of AMDPRDMP and works the same as a SUMMARY FORMAT control statement.
- GO — causes AMDPRDMP to execute a group of control statements, which may be provided with the ONGO statement, making it unnecessary for the user to specify each statement separately.
- GRSTRACE or QCBTRACE — causes AMDPRDMP to format and print the global and local resource queue control blocks in a dumped system.
- IOSDATA — causes AMDPRDMP to format and print the contents of certain I/O supervisor (IOS) control blocks in the input data set.
- JES3 — causes AMDPRDMP to format the contents of specific control blocks in the JES3 address space.

- LOGDATA — causes AMDPRDMP to invoke the formatter for the in-storage LOGREC buffer.
- LPAMAP — causes AMDPRDMP to format and print a map of the contents of the dumped system's link pack area directory entries (LPDE) and the active link pack area modules from the information in the contents directory entries (CDEs).
- MTRACE — causes AMDPRDMP to format and print the master trace table for the dumped system.
- NEWDUMP — defines a new input dump data set to AMDPRDMP.
- NEWTAPE — used instead of NEWDUMP to define a new tape input dump data set.
- NUCMAP — causes AMDPRDMP to format a map of the contents (name, entry point, entry point attributes, and length) of modules in the nucleus when the dump was generated.
- ONGO — specifies the control statements that will be executed when GO is issued.
- PRINT — causes AMDPRDMP to print and format particular areas of real and virtual storage.
- RSMDATA — causes AMDPRDMP to format and print the contents of the real storage manager control blocks in the input data set.
- SADMPMSG — causes AMDPRDMP to format and print the stand-alone dump console message log.
- SEGTAB — provides the hexadecimal real storage address of a segment table. If the user fails to do a store status prior to the execution of the stand-alone dump, the SEGTAB will not be found.
- SRMDATA — causes AMDPRDMP to format and print the contents of system resources manager control blocks (OUCBs and the Domain Table) that exist in the input data set.
- SUMDUMP — causes AMDPRDMP to invoke the formatter for summary dump data that an SVC dump might contain.
- SUMMARY — causes AMDPRDMP to print one of several reports, as specified by the parameters: FORMAT, KEYFIELD, JOBSUMMARY and TCBSUMMARY.
- TCAMMAP — causes AMDPRDMP to print selected TCAM control blocks.
- TITLE — provides AMDPRDMP with a title to be printed at the top of each page of the output listing.
- TRACE — causes AMDPRDMP to format trace entries in the system trace table.

- VSMDATA — causes AMDPRDMP to format and print the contents of the virtual storage manager control blocks in the input data set.
- VTAMMAP — causes AMDPRDMP to print selected VTAM control blocks.
- User-defined exit routines — cause AMDPRDMP to give control to a previously defined user or component exit module.

AMDPRDMP Output

Various formats may appear in the AMDPRDMP output listing, depending on which control statements the user selects. In addition to the formatted information, each page contains a heading that includes the optional user-specified title, the name of the module that invoked the dump (if the input data set was a dump data set), and the date and time that the dump was taken. The first page of the output lists the title given when the dump was taken. Within the formatted listing, comments may appear that are a result of invalid control blocks of AMDPRDMP's inability to locate, format, or print a control block.

After printing the formatted information, AMDPRDMP prints the following summary information:

- The number of entries to the read routine.
- The number of times that the required address was not found in a buffer.
- The number of blocks read from the dump data set.
- The number of permanent I/O errors encountered during execution.
- The average number of buffers used for each operation performed during execution.
- The number of GTF records read.
- The ratio of the number of times the read routine was called to the number of times the requested address was not in a buffer.

For a detailed description of the AMDPRDMP output listing, see *MVS/XA Debugging Handbook*.

Section 2: Method of Operation

This section describes how AMDPRDMP formats and prints information from an input data set. As shown in Figure 3-1, the description begins with an overview of AMDPRDMP, then describes the following stages of the overall processing:

- AMDPRDMP initialization.
- Control statement processing.
- Control statement execution.

Reading Method of Operation Diagrams

Method of operation diagrams are arranged in an input-processing-output layout: the left side of the diagram contains the data that serves as input to the processing steps in the center of the diagram, and the right side contains the data that is output from the processing steps. Each processing step is numbered; the number corresponds to the verbal description of the step in the extended description. Whereas the processing step in the diagram is in general terms, the corresponding text is a specific description that includes a cross-reference to the code for the processing.

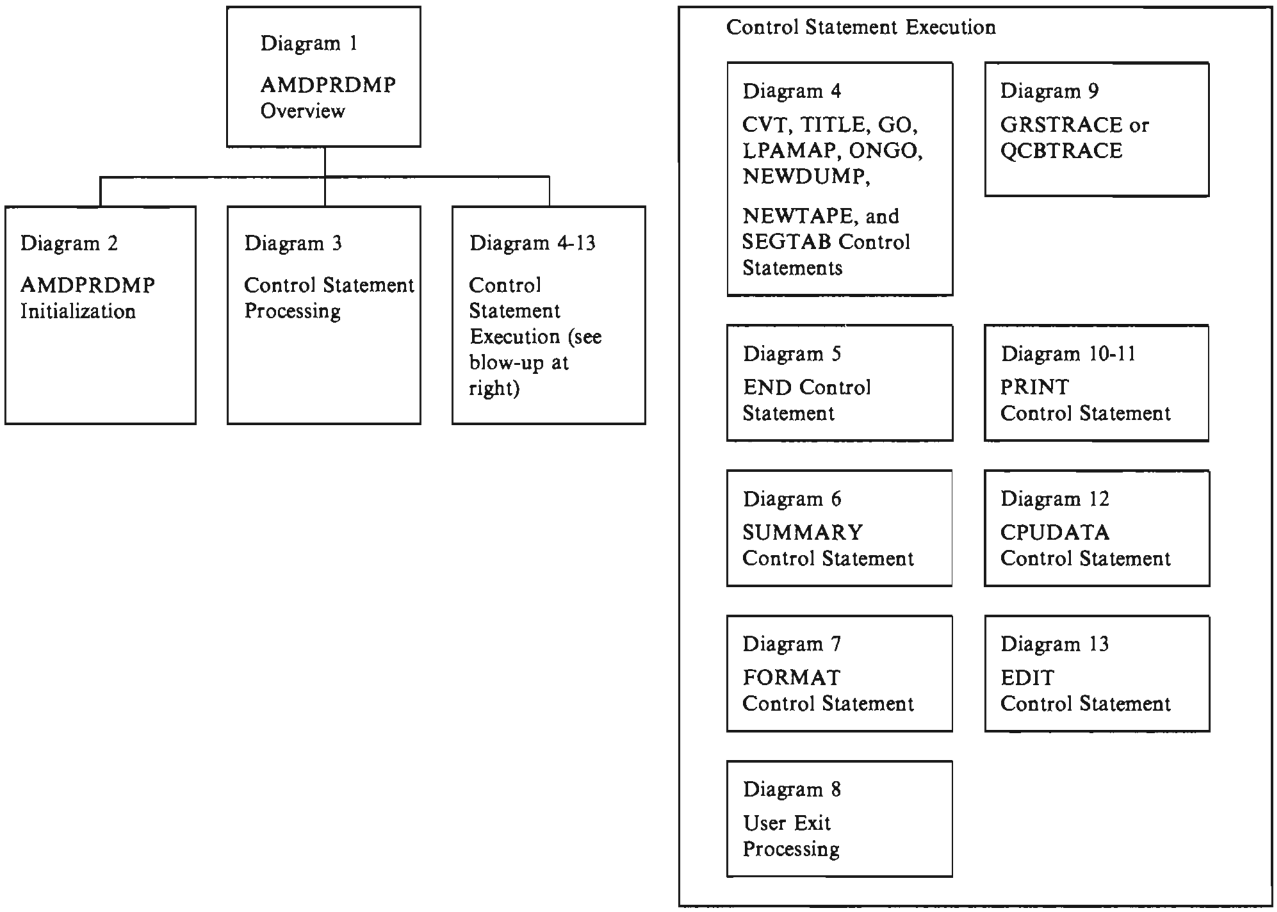


Figure 3-1 (Part 1 of 2). Key to Method of Operation Diagrams for AMDPRDMP

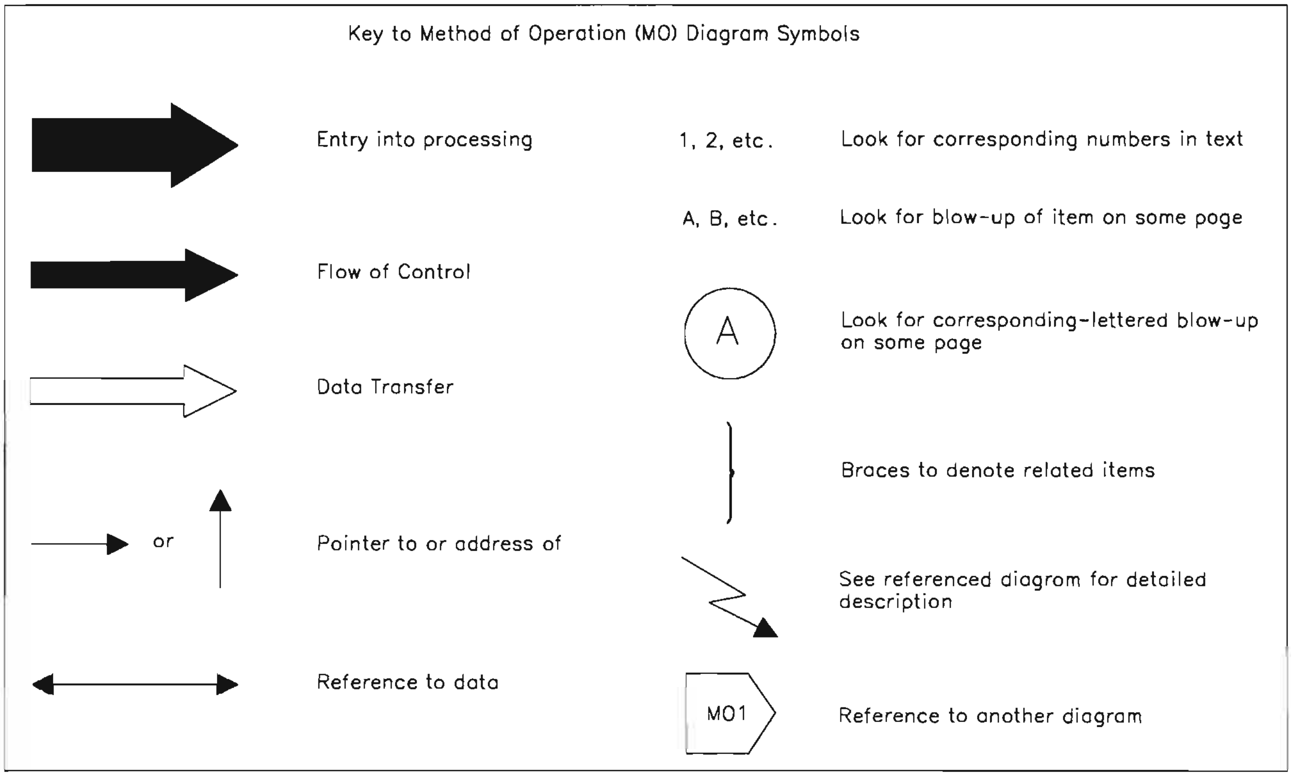
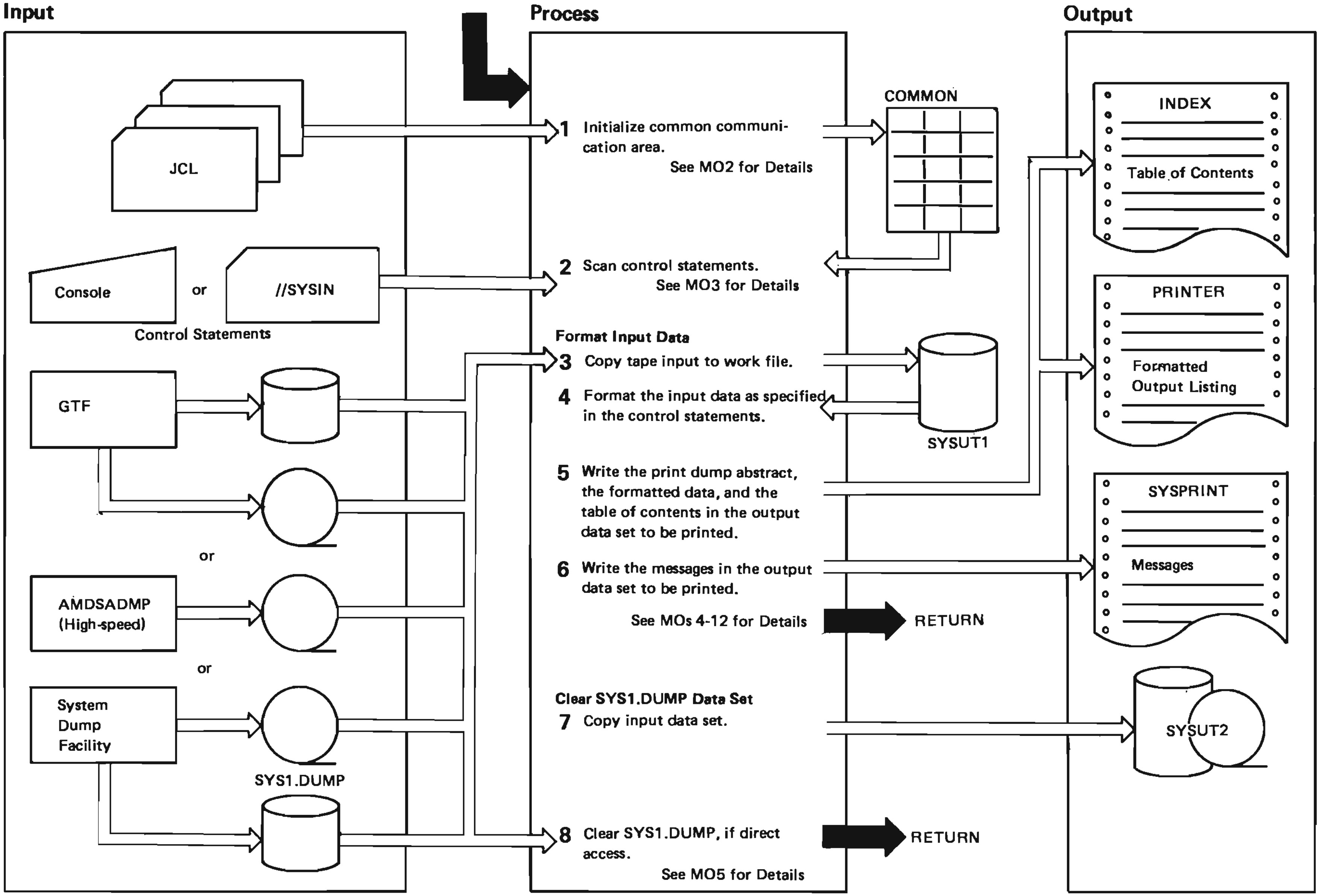


Figure 3-1 (Part 2 of 2). Key to Method of Operation Diagrams for AMDPRDMP

PRDMP Diagram 1. AMDPRDMP Overview (Part 2 of 2)

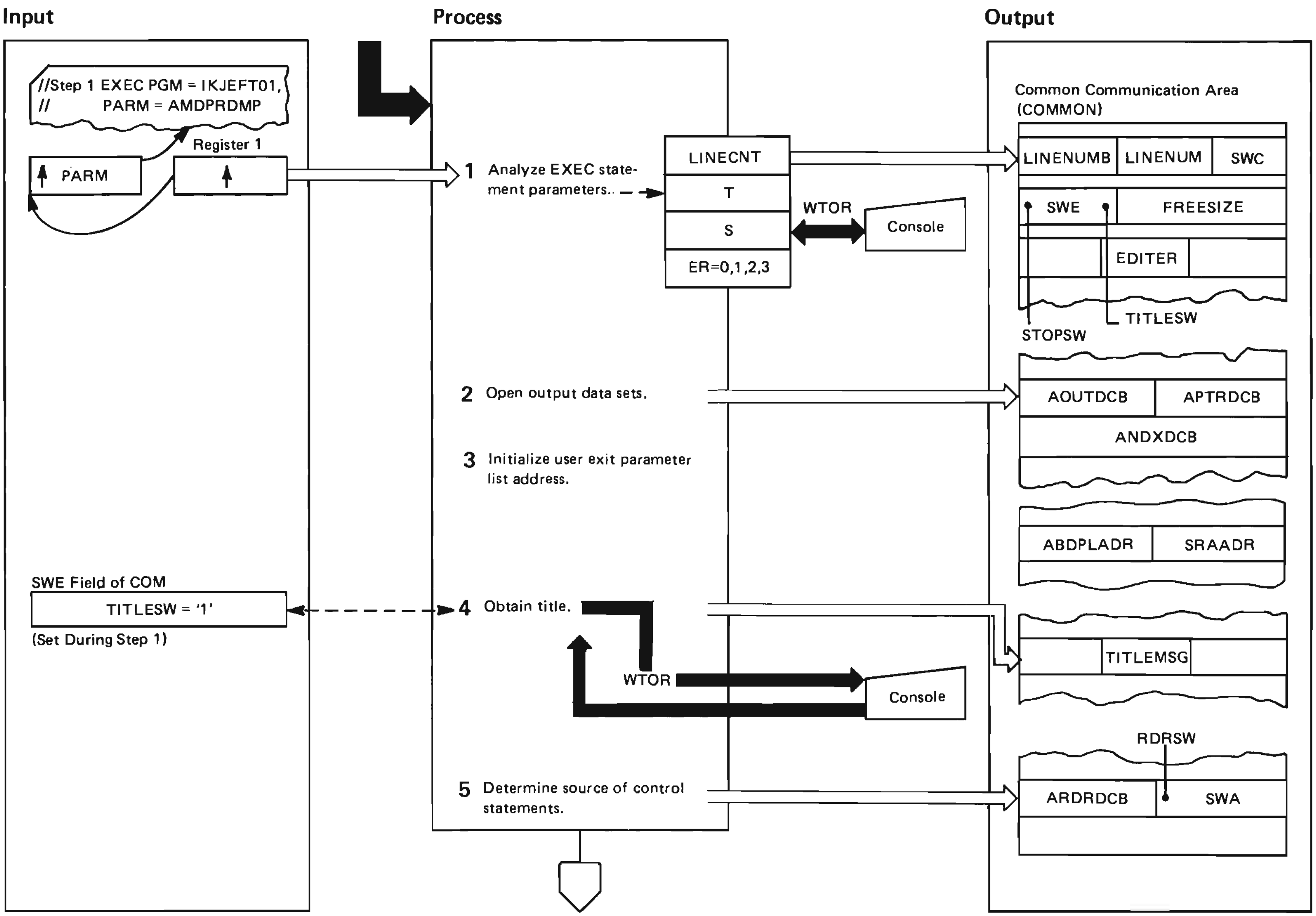


PRDMP Diagram 1. AMDPRDMP Overview (Part 2 of 2)

Extended Description	MO
1 The user controls part of the operation of AMDPRDMP with JCL statements. AMDPRDMP initializes the common communication area with values from the PARM field in the EXEC statement.	2
2 By using control statements, the user specifies the formatting to be performed by AMDPRDMP before the data is printed. The user enters the control statements into the system via either the input stream or the console. Depending on the function specified in the control statement, either EDIT modules, executor modules and their associated service modules, or user/component modules are brought into storage to perform the functions requested via the control statements.	3
3 If the user's JCL includes a SYSUT1 DD statement and if the input data set is on tape, AMDPRDMP copies the dump to the SYSUT1 data set. If SYSUT1 cannot contain the entire dump data set, AMDPRDMP processes the input from both SYSUT1 and tape. If the input is a SYS1.DUMP data set, AMDPRDMP copies the contents of the direct access input data set to the SYSUT1 data set.	
4 The input data to be formatted by AMDPRDMP is from the following sources: • The output data set from the generalized trace facility (GTF). • The output data set from the AMDSADMP service aid. • The SYS1.DUMP data set from the system dump facility, a SYSMDUMP ABEND dump, or any dumps produced by SVC dump. (The input data for all of these dumps appears the same to AMDPRDMP.) The information from these sources is formatted according to specifications in the control statements.	4-12

Extended Description	MO
5,6 AMDPRDMP writes the title pages, the formatted data, and messages in output data sets that will be printed for the user. If the INDEX DD statement is included in the JCL, the print dump table of contents is written to the INDEX data set.	4-12
7 If the user's JCL includes a SYSUT2 DD statement, AMDPRDMP transfers the contents of the dump data set to the SYSUT2 data set for later processing. All format control statements are ignored at this time.	5
8 AMDPRDMP clears the SYS1.DUMP data set so it can be reused. This processing occurs only if SYS1.DUMP is on direct access.	

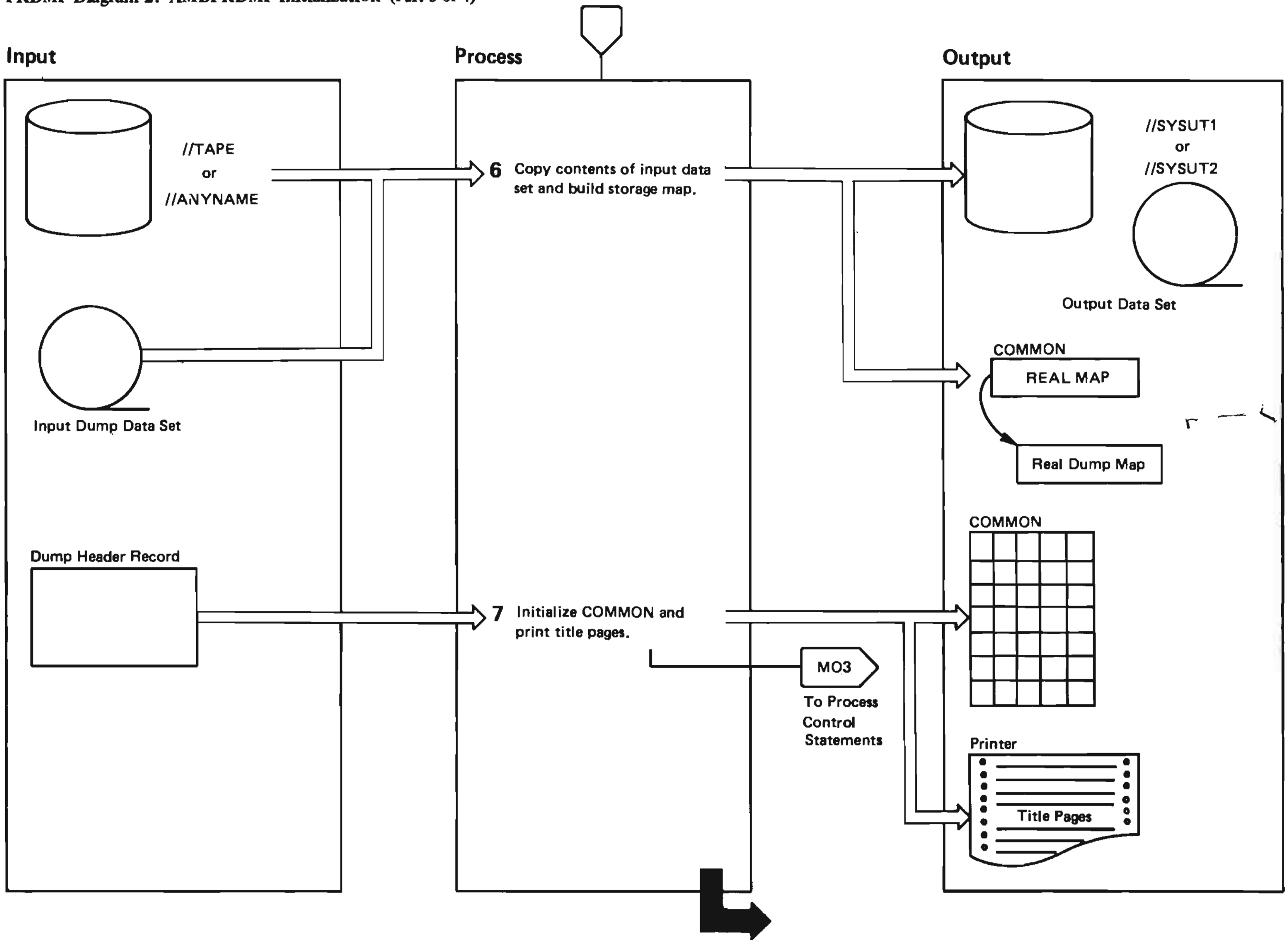
PRDMP Diagram 2. AMDPRDMP Initialization (Part 1 of 4)



PRDMP Diagram 2. AMDPRDMP Initialization (Part 2 of 4)

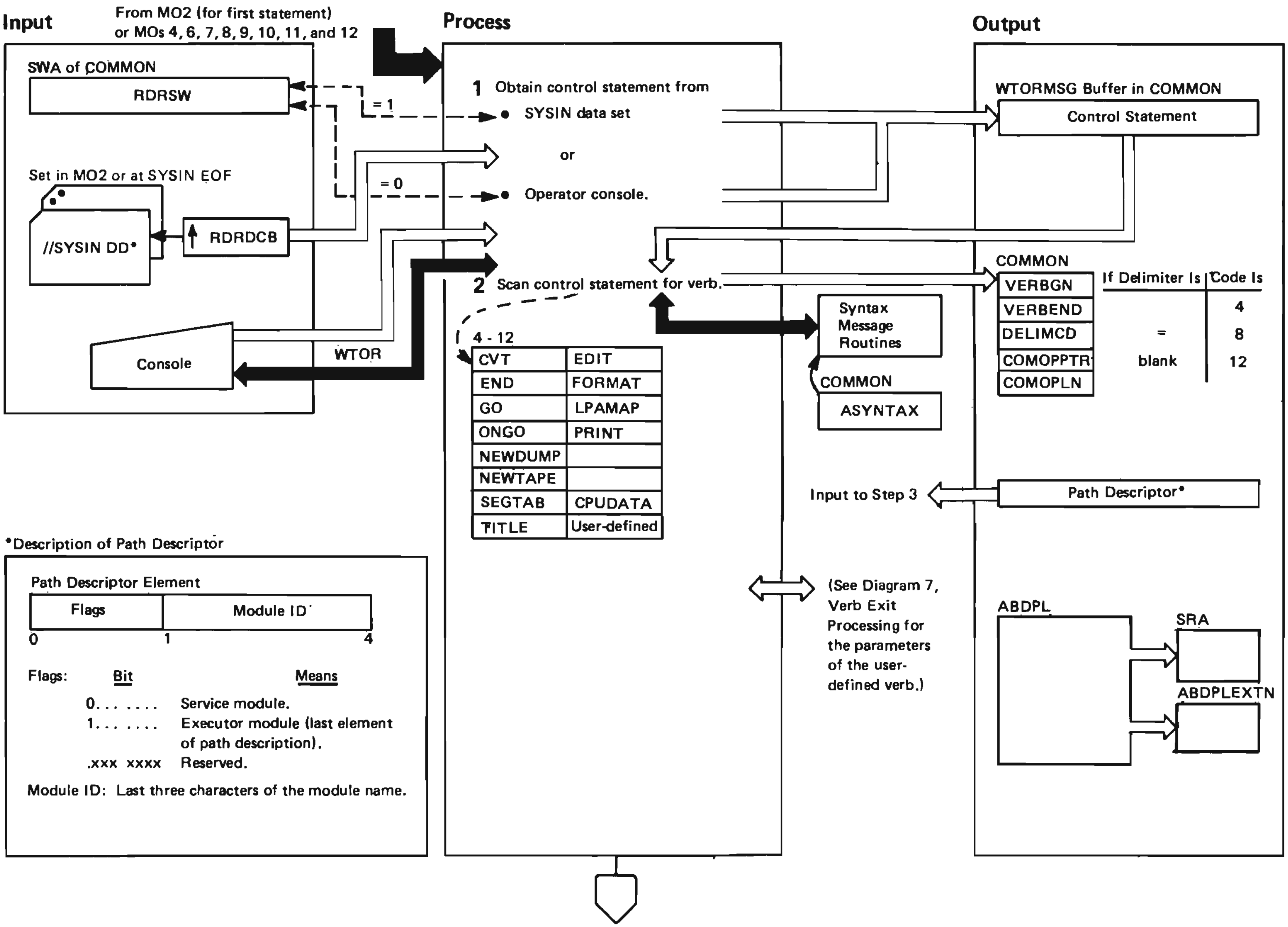
Explanation	Module	Label
1 The user may specify a list of parameters for AMDPRDMP with the operand PARM = 'AMDPRDMP, . . , . . .' in the EXEC statement. (A detailed discussion of the parameters and their meanings appears in *Service Aids.*)		
AMDPRDMP analyzes the parameters and places their values in the common communication area (COMMON) as follows:	AMDPRCTL	BGNLOOP
• LINECNT – If LINECNT is specified, AMDPRDMP stores the number of lines in the LINENUMB field if it is greater than 10 or less than 32K.	AMDPRCTL	BEGLCNT
• T – If the title option is specified, AMDPRDMP turns on bit 3 (TITLESW) of the SWE field. This bit is checked later (see step 4).	AMDPRCTL	BEGTITLE
• S – If the stop option is specified, AMDPRDMP turns on bit 0 (STOPSW) of the SWE field, then issues message AMD156I to the operator via a WTOR macro instruction. However, AMDPRDMP does not wait for a reply. At any time, the operator may reply with a STOP, and processing of the current data set will cease.	AMDPRCTL AMDPRCOM AMDPRCOM	BEGIN5 STPWTORM
• ER – If an action is specified for an error during execution of the EDIT function (see MO 12), AMDPRDMP stores the value in the EDITER field.	AMDPRCTL	BEGER
2 There are three possible output data sets. PRINTER contains the formatted output information, SYSPRINT contains the messages, and INDEX contains the dump table of contents. AMDPRDMP stores the address of PRINTER DCB in the AOUTDCB field of COMMON, the address of SYSPRINT DCB in the APTRDCB field, and the address of INDEX DCB in ANDXDCB after it opens the data sets.	AMDPRCOM AMDPRCOM AMDPRCOM	
3 AMDPRDMP puts the address of the ABDPL and the SRA in COMMON and initializes the parameter list for BLSQEXTI.	AMDPRUIM	AMDPRUIM
4 AMDPRDMP checks the TITLESW (bit 3) of SWE to determine whether the title option has been selected by the user. If so, AMDPRDMP issues message AMD154D to the operator via a WTOR macro instruction. The operator replies with the title.	AMDPRCTL AMDPRCTL	WRTMSG DUMPMSG
5 AMDPRDMP tries to open the reader DCB. If it opens, then there are control statements in the form of cards in the input stream. AMDPRDMP stores the address of the reader DCB in COMMON and turns on bit 7 (RDRSW) of SWA to indicate the presence of control cards. If the reader DCB does not open, the control statements must be obtained from the console.	AMDPRCTL AMDPRCTL	BEGIN6E BEGIN7

PRDMP Diagram 2. AMDPRDMP Initialization (Part 3 of 4)



Extended Description	Module	Label
6 If the input is from a tape or a direct access SYS1.DUMP data set, AMDPRDMP copies the input data set to the work-file data set and builds storage maps of the dump. If a work-file data set is not specified, AMDPRDMP builds the dump storage maps.	AMDPRLOD AMDPREAD	AMDPRLOD AMDPREAD
If a SYSUT2 data set is specified, AMDPRDMP copies the input data set to the SYSUT2 data set without building any storage maps.	AMDPRLOD	AMDPRLOD
7 AMDPRDMP initializes the common communication area using values from the dump header record.	AMDPRMST AMDPRCMC	AMDPRMST AMDPRCMC
AMDPRDMP writes the first title page to the print data set.	AMDPRMST	AMDPRMST
AMDPRDMP invokes all header exits including the DAE header exit.	AMDPRUIM	FMTABSRH
AMDPRDMP formats the abstract title page and the console-initiated loop trace records.	AMDPRABS AMDPRLRF	AMDPRABS AMDPRLRF

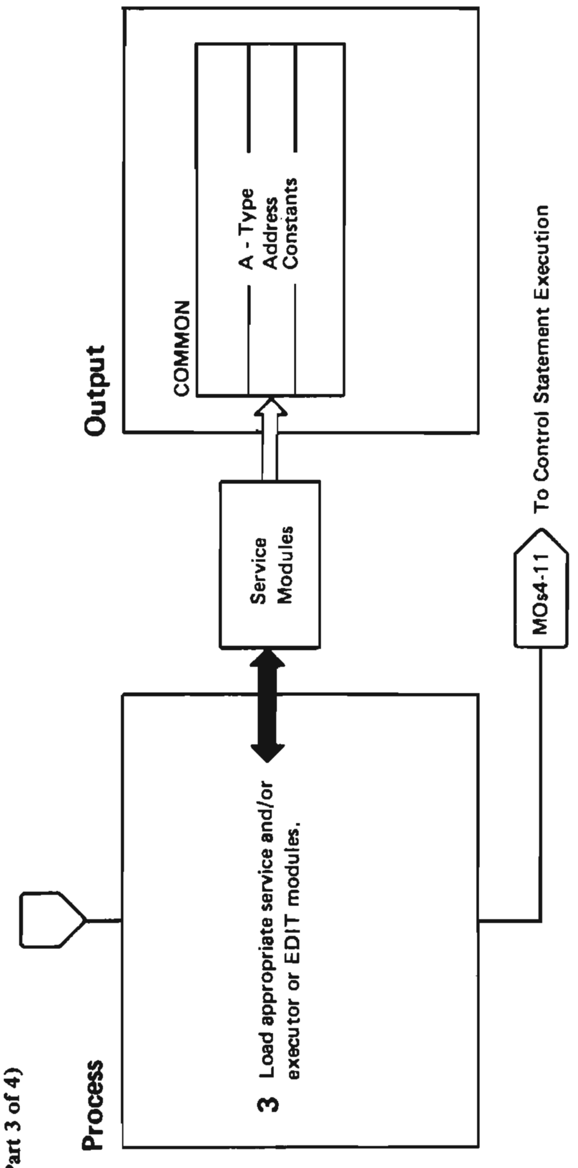
PRDMP Diagram 3. Control Statement Processing (Part 1 of 4)



Explanation	Module	Label
1 AMDPRDMP checks the RDRSW bit to determine whether there are control statements in the SYSIN date set. If the bit is on, AMDPRDMP reads one control card. If an end-of-file condition exists, AMDPRDMP issues message AMD1701 indicating end-of-file and sets RDRSW to 0. Then, or if the RDRSW bit is off, AMDPRDMP issues message AMD155D for the operator to enter a control statement. The control statements are stored one at a time in the WTORMSG buffer in COMMON.	AMDPRCTL	TRDRSW RDCARD RDREOF WRTMSG1
2 AMDPRDMP scans the WTORMSG buffer for the verb in the control statement. It saves the address of the beginning and end of the verb in the common area. It checks for the delimiter and stores a code indicating the delimiter in the DELIMCD field of COMMON. Finally, it compares the verb in the buffer to the valid verbs (listed above). If the verb is a user-defined verb, AMDPRDMP scans for parameters. If parameters exist, AMDPRDMP moves them (including parameters continued on subsequent input records) to a buffer and places the address of the buffer in COMOPPTR and the length of the parameters list in COMOPLN.	AMDPRCTL	VERBSCN5 VERBSCN3 VERBSCN4 EDIT OPERANDS
If the verb is PRINT, AMDPRDMP scans the statement beyond the verb level to determine the parameters.		PRINT
If there are syntax errors in the control statement, AMDPRDMP writes a message in the SYSPRINT data set.	AMDPRCOM	SYNTAX/ SYNTAXA-E
However, for the user and other component defined verbs, AMDPRUIM scans the ECT (exit control table) to determine if the verb is a valid exit. If the verb is a user or component exit, control is passed to the appropriate module.	AMDPRUIM	User or other component defined verb exit.
AMDPRUIM initializes the user exit interface. The ABDPL, its extension and the SRA are reinitialized for each exit invocation. A GETMAIN issued will obtain storage for an operands buffer. This buffer is addressed by the ABDPL extension in two ways. One manner acquires a direct pointer to operands.		

Explanation	Module	Label
The other uses a CPPL to obtain a standard TSO command buffer form. If the control statement was FORMAT, AMDPRUIM will translate it into a request for the SUMMARY verb exit with the FORMAT keyword. BLSQEXTI is called to load exit service routines and finish initialization of ABDPL and SRA. The Exit Services Router is called to invoke the ECT service, which will scan the ECT and link to the exit. (See IPCS Logic for a description of BLSQEXTI, the Exit Services Router, and the exit services.) (See Diagrams 7 and 8).		

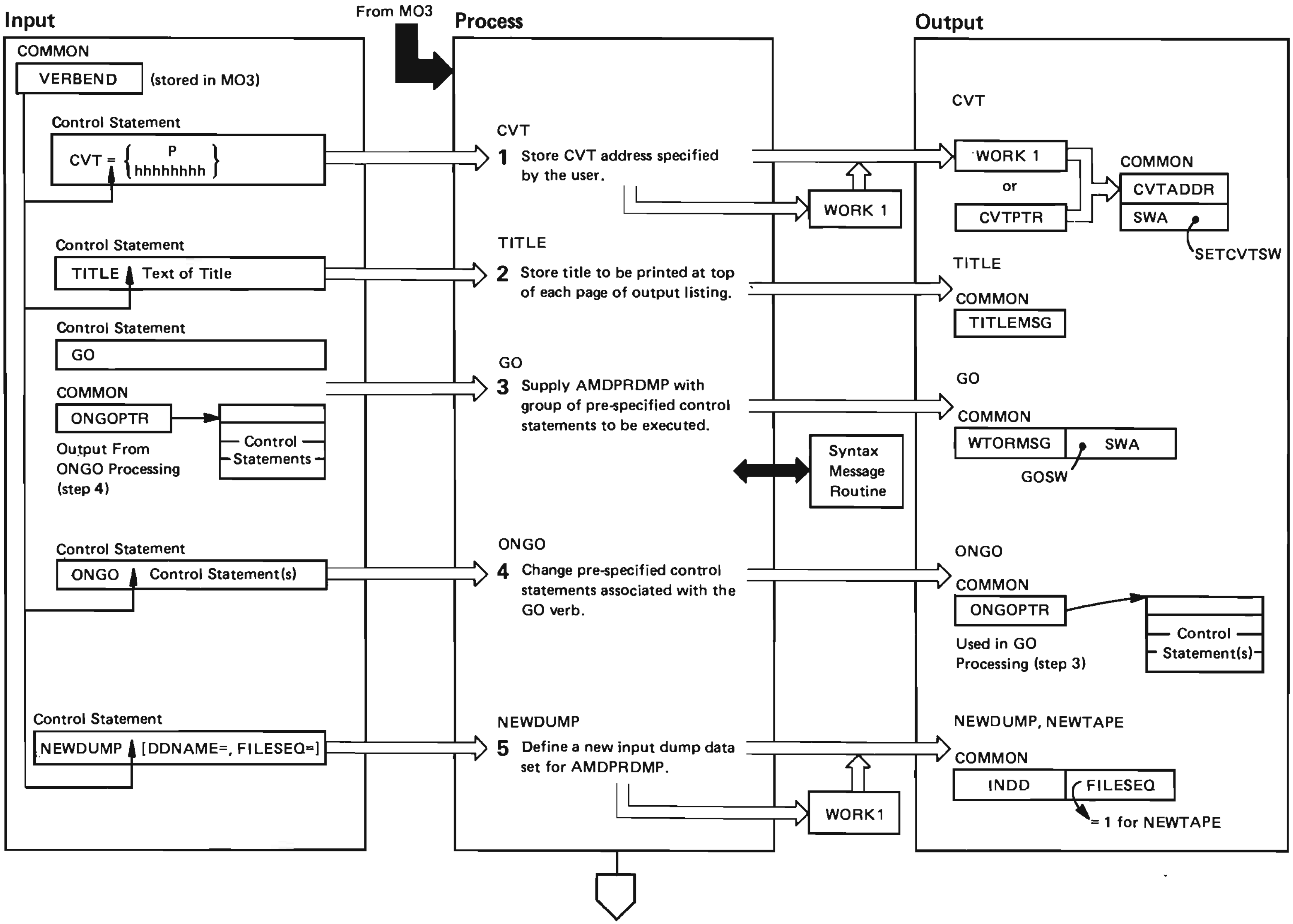
PRDMP Diagram 3. Control Statement Processing (Part 3 of 4)



PRDMP Diagram 3. Control Statement Processing (Part 4 of 4)

Explanation	Module	Label
3 The routines for executing function control statements are resident in virtual storage, and the function is executed without further processing.	AMDPRCTL	CVTSET TITLEVRB NEWDUMP END GO ONGO NEWTDUMP SEGTBSET
To execute format control statements, AMDPRDMP loads the appropriate path of service and/or executor modules shown in Figure 2 or EDIT modules shown in Figure 3. (The service modules perform functions that are common to several executor modules. The executor modules govern the execution of the functions specified in the control statement.) To load the path, AMDPRDMP uses the path descriptor for the modules. AMDPRDMP compares the path descriptor with a list of modules that have been loaded previously. Previous modules not required by this control statement are deleted, and the required modules are loaded. During the load, control is given to each service module to store entry points in COMMON. The last module in each path that is loaded is the executor module.	AMDPRCTL	 CPUDATA PRNTFMT PRINT EDIT
	AMDPRSEG	LOAD
	AMDPRFUB	
	AMDPRDPS	DPBASE

PRDMP Diagram 4. The CVT, TITLE, GO, ONGO, NEWDUMP, NEWTAPE, SEGTAB, and LPAMAP Control Statements (Part 1 of 4)



PRDMP Diagram 4. The CVT, TITLE, GO, ONGO, NEWDUMP, NEWTAPE, SEGTAB, and LPAMAP Control Statements (Part 2 of 4)

Extended Description	Module	Label
1 CVT — If the user believes that the communications vector table (CVT) pointer at storage location X'4C' has been destroyed, he supplies AMDPRDMP with the CVT address in the CVT control statement. AMDPRDMP finds the parameter via the address in the VERBEND field of COMMON, then checks the parameter for syntax errors. Then AMDPRDMP determines what address is specified. If P is specified, AMDPRDMP stores the address if the location at X'4C' in the CVTADDR field of COMMON. If a hexadecimal address is specified, AMDPRDMP converts the address to binary, then stores it in the CVTADDR field. Finally, it indicates that the CVT address has been stored by turning on bit 6 (SETCVTSW) in SWA.	AMDPRCTL	CVTSET CVTSETA CHECKCVT CVTOK CVTSET2
2 TITLE — The user may specify the title to be printed at the top of each page via the title control statement. When a title control statement is encountered, AMDPRDMP locates the parameter via the address in the VERBEND field, then stores the title in the TITLEMSG field of COMMON.	AMDPRCTL	TITLEVRB TITLMOVE
3 GO — By issuing the GO control statement the user causes AMDPRDMP to format the input data as if the user had specified the EDIT, SUMMARY, and PRINT CURRENT control statements. By using the ONGO control statement (see step 4), the user may change the group of pre-specified control statements to any combination of CVT, GRSTRACE, or QCBTRACE, LPAMAP, FORMAT, PRINT (with its parameters), SEGTAB, SUMMARY, CVTMAP, CPUDATA, SUMDUMP, SRMDATA, ASMDATA, LOGDATA, VTAMMAP, or EDIT. When AMDPRDMP encounters a GO control statement, the program uses the address in the ONGOPTR field of COMMON to obtain the parameter. The parameter will be the pre-specified default control statements or, if ONGO was used, any control statements. AMDPRDMP stores the parameter in the WTORMSG buffer of COMMON, turns on bit 4 (GOSW) in SWA, and passes the parameter to be scanned (see MO3) and executed as control statements.	AMDPRCTL	GO GOONGO GO VERBSCN5

Extended Description	Module	Label
4 ONGO — By using the ONGO control statement, the user changes the group of control statements associated with GO (see step 3). When AMDPRDMP encounters an ONGO control statement, it locates the parameter via the address in the VERBEND field of COMMON, stores the parameter, and places its address in the ONGOPTB field of COMMON. When GO is used, AMDPRDMP will execute the new set of control statements.	AMDPRCTL	ONGO ONGO1 ONGO2 GO
5 NEWDUMP — By using the NEWDUMP control statement, the user defines a new input dump data set for AMDPRDMP. Two keyword parameters may be specified:		
• IF DDNAME is specified, AMDPRDMP stores the DDNAME in the INDD field of COMMON. The DDNAME identifies the DD statement that describes the input dump data set.	AMDPRCTL	NEWDD
• If FILESEQ is specified, AMDPRDMP isolates the value, converts it to binary in WORK1 buffer of COMMON, then stores it in the FILESEQ field.		NEWFILE
If neither keyword is specified, AMDPRDMP assumes that the new input dump data set is the same as for NEWTAPE (see step 6). When NEWDUMP is specified, AMDPRDMP resets switches associated with the former dump; it also resets the CVT and SEGTAB addresses and the address of the top of real storage.	AMDPRCTL	NEWTDUMP NEWRESET

PRDMP Diagram 4. The CVT, TITLE, GO, ONGO, NEWDUMP, NEWTAPE, SEGTAB, and LPAMAP Control Statements (Part 3 of 4)

Input

COMMON
VERBEND

Control Statement
NEWTAPE

Control Statement
SEGTAB= hhhhhhhh

Control Statement
LPAMAP [EPA] [,MODNAME]

CVT
CVTLPDIA

LPDEs
Module Name
EPA
LPDE Attributes
Major LPDE Name
Module Length
Module Location
(Second Entry)
⋮

CVT
CVTQLPAQ
First CDE

CDEs
CDE Attributes
Module Name
EPA

Extent List
Length of CSECT
CSECT

Process

NEWTAPE

6 Define a new tape input dump data set for AMDPRDMP.

SEGTAB

7 Store segment table address specified by user; set page size.

LPAMAP

8 Format and print a map of the link pack area directory entries in the link pack area active queue.

If CVTLPDIA address is not found, or if no reenterable load module queue exists, issue message.

MO3 To Get Next Control Statement

Output

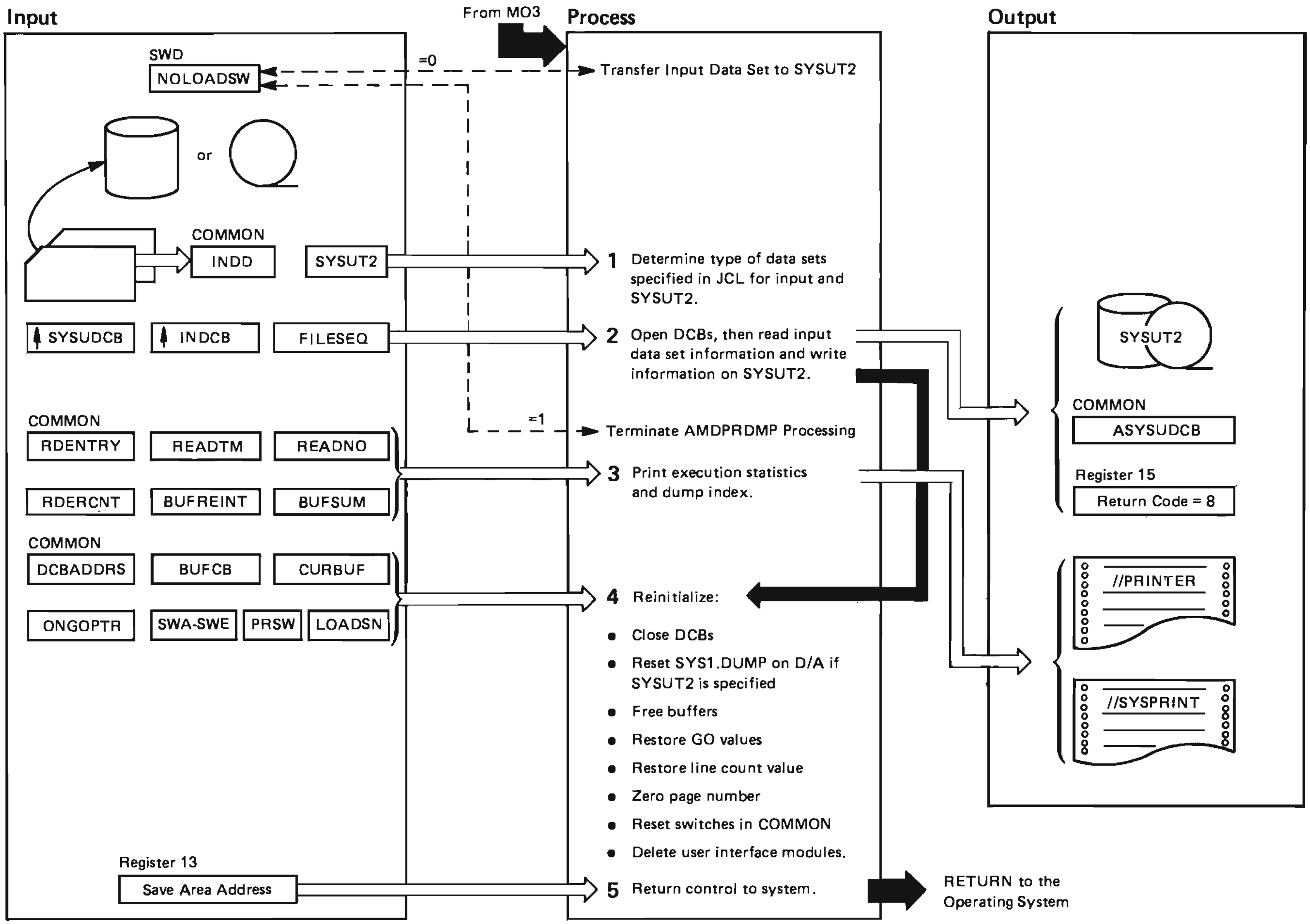
SWA | SWC | CVTADDR
SETCVTSW=0
MSTRSW=0

COMMON
SEGTABOR | PRSW
QSEGTBSW

// PRINTER

Extended Description	Module	Label
6 NEWTAPE — The NEWTAPE control statement is used in place of NEWDUMP to define new tape input. There are no keyword parameters associated with NEWTAPE, so AMDPRDMP assumes the TAPE DD statement, and FILESEQ=1. It stores the assumed values in the INDD, and FILESEQ fields of COMMON and resets switches from the former dump as it does for NEWDUMP (see step 5).	AMDPRCTL	NEWTDUMP NEWRESET
7 SEGTAB — If the user forgets to perform a store status operation prior to executing a stand-alone dump (AMDSADMP), the user supplies AMDPRDMP with the segment table address in the SEGTAB control statement. AMDPRDMP locates the parameter via the address in the VERBEND field of COMMON, then stores the SEGTAB address in the SEGTABOR field. Finally, bit 1 (QSEGTBSW) of PRSW is set to indicate the segment table address has been stored	AMDPRCTL	SEGTBSET
8 LPAMAP — The LPAMAP control statement causes AMDPRDMP to format a list of the modules in the link pack area directory entries end the modules in the active link pack area queue. LPAMAP then causes AMDPRDMP to write the formatted information in the PRINTER output data set to be printed in two columns in the output listing.		
AMDPRDMP obtains information about the module names, the entry point addresses (EPAs) in the link pack area of the link pack directory entry (LPDE), and the EPAs of the contents directory entry (CDE). AMDPRDMP locates the address of the first LPDE in the CVT at CVTLPDIA and the address of the first CDE on the active queue in the CVT at CVTQLPAQ. If AMDPRDMP finds a minor entry, it obtains the length and the starting eddress of the module's CSECTs from the LPDE and the CDE. AMDPRDMP writes all of the information in the PRINTER data set to be printed on a separate page of the AMDPRDMP output listing.	AMDPRLPA	AMDPRLPA

PRDMP Diagram 5. The END Control Statement (Part 1 of 2)



PRDMP Diagram 5. The END Control Statement (Part 2 of 2)

Extended Description	Module	Label
When the control statement is END, AMDPRDMP checks the NOLOADSW switch in COMMON to determine whether END is the only control statement for this execution of AMDPRDMP. If NOLOADSW is off, END is the only control statement for this execution, and the purpose of the execution is to transfer the contents of an input data set to the SYSUT2 data for later processing by AMDPRDMP. If the data set is SYS1.DUMP on direct access, the data set is cleared to allow for more dumps. If NOLOADSW is on, there were other control statements in this execution of AMDPRDMP, and END terminates this execution.	AMDPRCTL	END
1 When END is used to transfer a data set, AMDPRDMP uses a DEVTYPE macro instruction to determine the type of input and SYSUT2 data set that are specified in the JCL. They may be either tape or direct access volumes.	AMDPRRDC	TSTDEV
2 The work data set load module, AMDPRLOD is loaded to read the specified input data set and write the data in the SYSUT2 data set. When the transfer is complete, AMDPRLOD requests termination of AMDPRDMP by returning a code of 8 in register 15.	AMDPRLOD	READNXT
3 If the NOLOADSW switch in SWD is on, END is not the only control statement in this execution of AMDPRDMP; the purpose of END in this case is to terminate AMDPRDMP. AMDPRDMP uses the information stored in COMMON to write execution statistics messages in the SYSPRINT data set. The statistics are not printed when END is used for a data set transfer. When the JCL does not include an INDEX DD statement, PRDMP prints the dump index at the end of the PRINTER data set. When the JCL includes an INDEX DD statement, the dump index is output to the INDEX data sat.	AMDPRCTL	ENDSTAT0 -ENDSTAT7
4 The END control statement terminates AMDPRDMP, both following a data set transfer and following other formatting operations. Termination processing includes: closing open DCBs and freeing associated buffer pools, freeing output and input buffers, restoring original GO values, freeing storage occupied by the ONGO buffer, restoring the default line count, zeroing the page number and resetting COMMON switches. AMDPRDMP deletes the AMDPRFMT and AMDPRECT modules.	AMDPRCTL AMDPRUIM	ENLOOP2 AMDUSRDL
AMDPRDMP calls PLSQEXT1 to delete the exit services it loaded in, since they are not required for the control statement execution. (Refer to Diagram 3, Step 3 and the explanation listed in the Control Statement Processing under Part 3).		
5 Finally, AMDPRDMP locates the highest level save area address and returns control to the operating system.	AMDPRCTL	ENDALL

PRDMP Diagram 7. Verb Exit Processing Table (Part 1 of 1)

Note: All input is via Register 1, which contains the address of the verb exit parameter list, BLSABDPL. Specific modules utilize PRDMP service routines and are responsible for the processing of the control blocks to produce a printed dump data set.

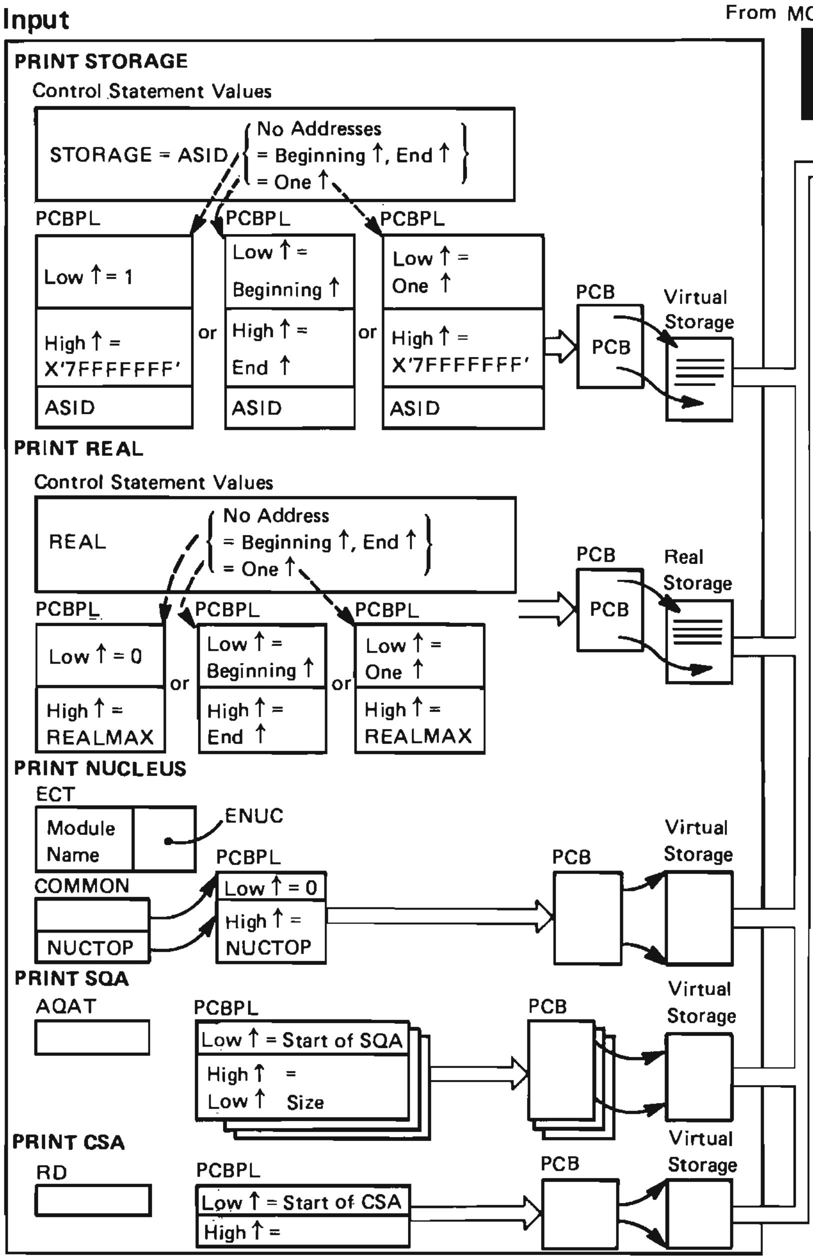
Verb exits are control statements that represent program modules. They are defined in the exit control table (ECT). AMDPRDMP checks the ECT and passes control to these modules when the corresponding control statement is specified.

Input	Process	Module
ASMDATA	The ASMDATA control statement causes AMDPRDMP to format and print the contents of the auxiliary storage manager control blocks from the input data set. For a list of the control blocks printed, see *MVS/XA Service Aids.* For a description of ILRFTMAN, see *MVS/XA System Logic Library.*	ILRFTMAN
CVTMAP	The CVTMAP control statement causes AMDPRDMP to format the communications vector table and write the formatted information in the PRINTER data set for printing.	AMDPRCVT
DAEDATA	The DAEDATA control statement causes AMDPRDMP to invoke ADYHDFMT, the DAE format routine. ADYHDFMT formats and prints information from the dump header record. For a description of ADYHDFMT output, see the *MVS/XA Debugging Handbook.*	ADYHDFMT
FORMAT	The FORMAT control statement causes PRDMP, via AMDPRUIM and BLSQSUM1, to format and print the major system control blocks associated with each address space. AMDPRUIM transposes the FORMAT control statement into SUMMARY FORMAT and invokes the exit services router to process the SUMMARY verb exit. For a description of BLSQSUM1, see *MVS/XA IPCS Logic.*	AMDPRUIM
GRSTRACE	The GRSTRACE, Q, or QCBTRACE control statement causes PRDMP to format and print the global or local resource queues in the input data set. For a description of the control blocks printed, see *MVS/XA SPL: Service Aids.* For a description of IOSGDPDMP, see *Global Resource Serialization Logic.*	ISGDPDMP
IOSDATA	The IOSDATA control statement causes AMDPRDMP to format and print the contents of specific I/O supervisor (IOS) control blocks from the input data set. For a list of the control blocks printed, see *MVS/XA SPL: Service Aids.* For a description of IOSVFMTH, see *MVS/XA System Logic Library, Volume 6, Part 2.*	IOSVFMT
JES3	The JES3 control statement causes AMDPRDMP to format contents of specific control blocks within the JES3 address space.	IATABPR
LOGDATA	The LOGDATA control statement causes AMDPRDMP to invoke IFCERFMT via the user exit interface. IFCERFMT formats and prints the records in the in-storage LOGREC buffer at the time of the dump.	IFCERFMT
MTRACE	The MTRACE control statement causes AMDPRDMP to format and print the contents of the master trace table for the dumped system in a first-in, first-out order. *MVS/XA Diagnostic Techniques* describes this table. *MVS/XA System Logic Library, Volume 6, Part 2* describes IEEMB817.	IEEMB817
NUCMAP	The NUCMAP control statement causes AMDPRDMP to format and print the contents of the nucleus map entries in the dump data set. See *MVS/XA SPL: Service Aids* for more information.	IEAVNUCM
RSMDATA	The RSMDATA control statement causes AMDPRDMP to format and print the contents of the real storage manager control blocks from the input data set. For a list of the control blocks printed, see *MVS/XA SPL: Service Aids.*	IARRDMP
SADMPMSG	The SADMPMSG control statement causes AMDPRDMP to format and print the stand-alone dump console message log. See *MVS/XA SPL: Service Aids.*	AMDSAFCM
SRMDATA	The SRMDATA control statement causes AMDPRDMP to format and print the OUCBs (system resource manager user control block) on the WAIT, OUT, and IN queues and the DMDT (domain descriptor table).	IRARMFMT

SUMDUMP	The SUMDUMP control statement causes AMDPRDMP to pass control to the summary dump format routine (IEAVTFSD) via the user exit interface. IEAVTFSD is described with the summary dump function of SVC dump under the recovery termination management (RTM) component in the *MVS/XA System Logic Library, Volume 6, Part 2* .	IEAVTFSD
SUMMARY	The SUMMARY control statement causes PRDMP to format and print a job summary, a TCB summary and/or major system control blocks in full or key-field form. PRDMP accesses BLSQSUM1 to complete the task. For a description of BLSQSUM1, see *MVS/XA IPCS Logic*.	BLSQSUM1
TCAMMAP	The TCAMMAP control statement causes AMDPRDMP to print ACF/TCAM control blocks that are helpful in ACF/TCAM problem determination. AMDPRDMP locates and formats selected ACF/TCAM control blocks in dumps produced by the AMDSADMP and SVC dump programs. For a description of the ACF/TCAM formatted dump routine and the control blocks selected, see *ACF/TCAM Diagnosis Guide*.	IEDPRDMP
TRACE	The TRACE control statement causes AMDPRDMP to pass control to the system trace formatter controller routine (IEAVETFC) to format and print the system trace table. For a description of the system trace table, see *MVS/XA Diagnostic Techniques*. For a description of the TRACE control statement, see *MVS/XA SPL: Service Aids*.	IEAVETFC
VSMDATA	The VSMDATA control statement causes AMDPRDMP to format and print the contents of the virtual storage manager control blocks from the input data set. These control blocks are described in *MVS/XA SPL: Service Aids*. For a description of IGVSFMAN, see *MVS/XA System Logic Library, Volume 6, Part 2*.	IGVSFMAN
VTAMMAP	The VTAMMAP control statement causes AMDPRDMP to print ACF/VTAM control blocks that are helpful in ACF/VTAM problem determination. AMDPRDMP locates and formats selected ACF/VTAM control blocks in dumps produced by the AMDSADMP and SVC dump programs. For a description of the ACF/VTAM formatted dump routine and the control blocks selected, see *ACF/VTAM Diagnosis Guide*.	ISTRAFD1

PRDMP Diagram 9. The PRINT STORAGE/REAL/NUCLEUS/SQA/CSA Control Statements (Part 1 of 2)

Input



Process

PRINT CONTROL STATEMENT

PRINT STORAGE/PRINT REAL/ PRINT NUCLEUS/PRINT CSA/PRINT SQA

1 Build and enqueue PCBs to print requested storage or pages:

2 • PRINT STORAGE — Print virtual storage range for job as indicated by control statement. If no address space is specified AMDPRDMP will attempt to process the four cross memory address spaces. If the cross memory address spaces are not in the dump data set, or if AMDPRDMP cannot identify them, the address space active on the processor on which the dump was taken will be processed.

• PRINT REAL — Print real storage range as indicated.

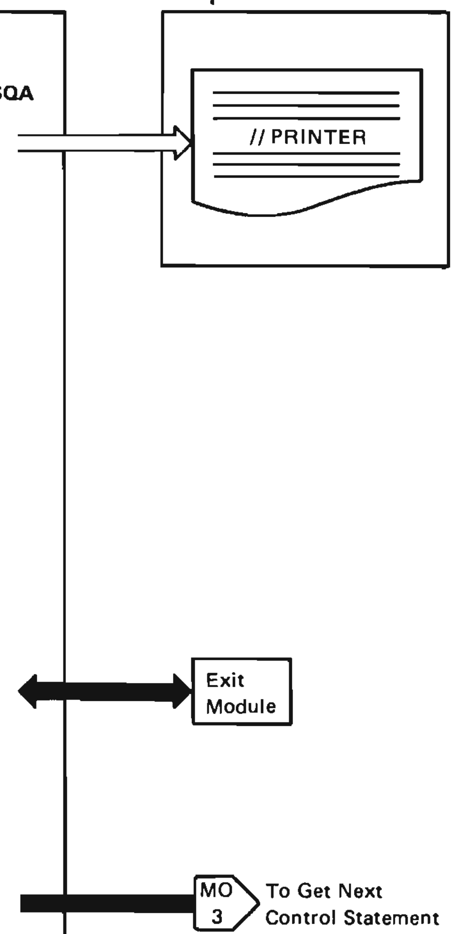
• PRINT NUCLEUS — Print DAT-on and DAT-off nucleus.

• PRINT CSA — Print common service area.

• PRINT SQA — Print system queue area.

3 PRINT REAL/STORAGE — Print registers and current PSW of IPL'ed CPU.

Output



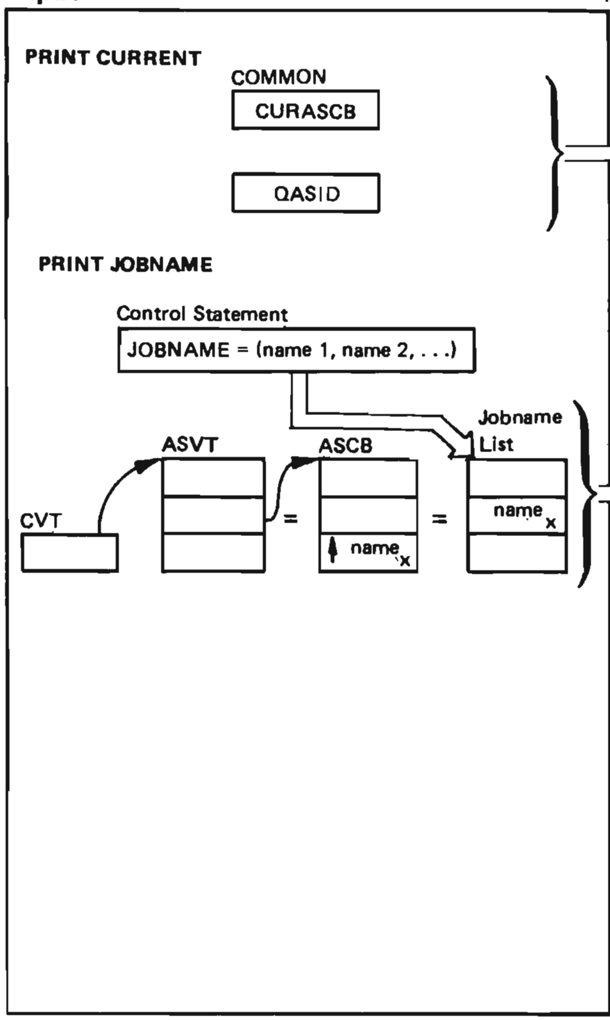
PRDMP Diagram 9. The PRINT STORAGE/REAL/NUCLEUS/SQA/CSA Control Statements (Part 2 of 2)

Extended Description	Module	Label
The PRINT control statement, along with its parameters, causes AMDPRDMP to format and print areas of real or virtual storage. During initialization, AMDPRDMP stores information describing the storage map used during PRINT execution in the print dump common communication area (COMMON).		
1 When AMDPRDMP encounters a PRINT control statement with any of the parameters listed below, it sets up a parameter list (PCBPL) describing the storage to be printed.		
2 • STORAGE – If the user supplied ASID(s), AMDPRDMP places the value in PCBPL. If no address space is specified AMDPRDMP attempts to process the four cross memory address spaces (home, primary, secondary, cross memory lock). If any of the address spaces have the same ASID as another, the address space will only be processed once. If AMDPRDMP is unable to determine the cross memory address spaces, or if they are not in the input data set, the default is the ASID of the job active on the CPU from which the dump was taken (from QASID in COMMON). The range of the address space to be printed is determined from the control statement. If only one address is specified, that address is used as the low address and the highest virtual storage address as the high address. If no address is specified, AMDPRDMP prints the entire address space. The printed output appears as follows: • Nonextended and extended private storage areas for each specified ASID in the dump • Nonextended COMMON • PSA (0-4K) and the entire DAT-on nucleus • Extended COMMON.	AMDPRPMS	
• REAL – AMDPRDMP sets up PCBPL with an address range for real storage. The high and low addresses are determined as for STORAGE.	AMDPRPMS	
• NUCLEUS – AMDPRDMP sets up a PCBPL for the DAT-on nucleus. If the DAT-off nucleus is contained in the dump, AMDPRDMP sets up a PCBPL for it. An SVC dump may not contain the DAT-off nucleus.	AMDPRNUC	
• CSA – Using the address ranges taken from the GDA during print dump initialization, AMDPRDMP sets up a PCBPL for both the nonextended and extended CSA.	AMDPRNUC	
• SQA – Using the address ranges taken from the AQAT index tables and the AQAT tables for each system queue area subpool, AMDPRDMP sets up a PCBPL for both the nonextended and extended SQA. If the address ranges are not available, AMDPRDMP sets up a PCBPL using the address ranges from the GDA.	AMDPRNUC	
3 For the STORAGE or REAL parameter (with no subparameters), AMDPRDMP writes the contents of the 16 general purpose registers, 16 control registers, 4 floating point registers, the prefix register, and the current PSW from the CPU on which the dump was taken. If input is from AMDSADMP, this information is taken from the CPUSTATUS record. Otherwise, it is from the dump header record.	AMDPRDPS	

PRDMP Diagram 10. The PRINT CURRENT/JOBNAME Control Statements (Part 1 of 4)

From MO3

Input



Process

1 Obtain ASCB addresses:

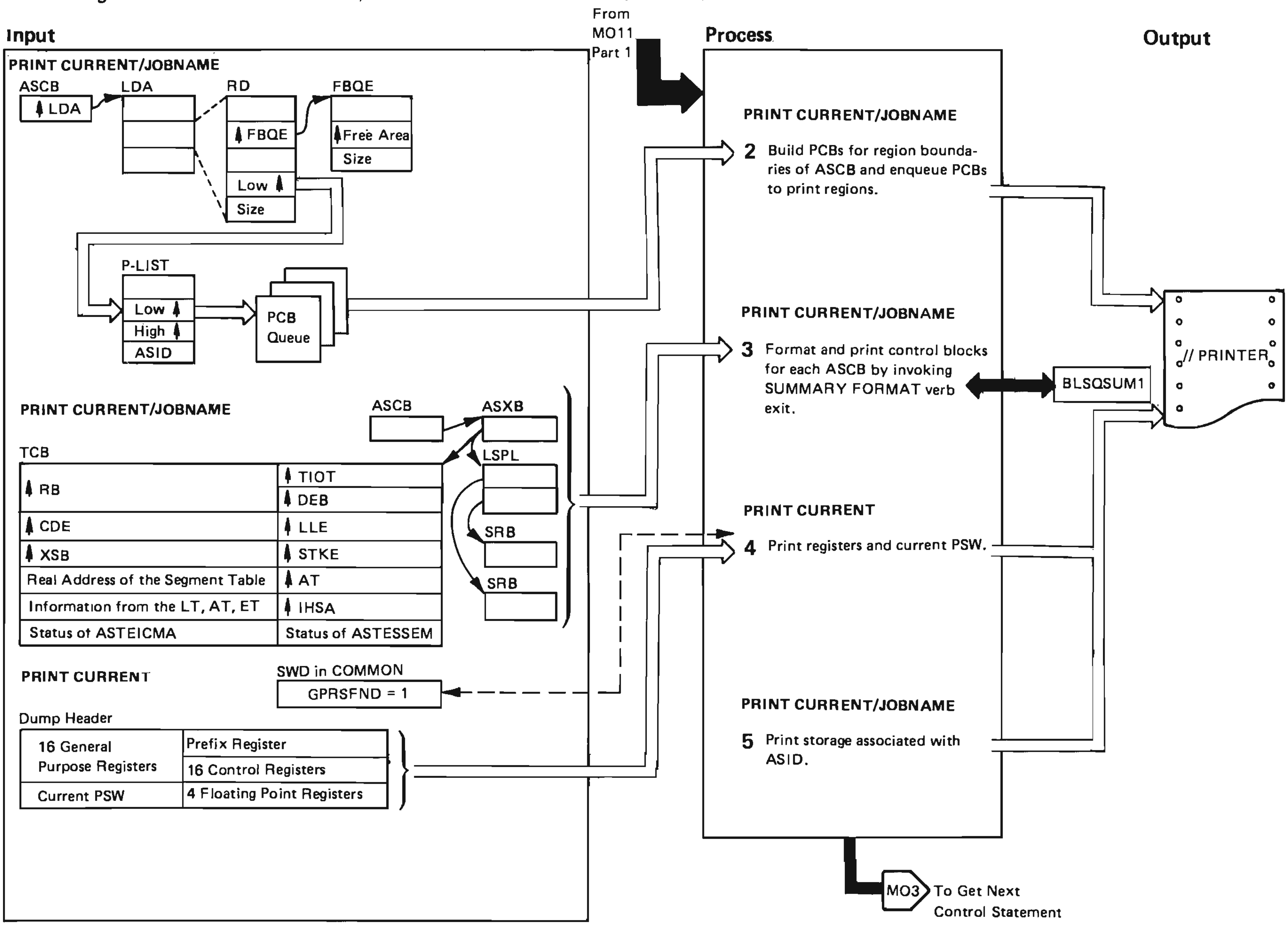
- PRINT CURRENT — For SVCDUMP or SYSMDUMP, AMDPRDMP obtains the cross memory address spaces from COMMON. For SADMP, AMDPRDMP obtains the cross memory address spaces for each online processor using the CPU status record, ASVT, and PSA. If any of the four cross memory address spaces (home, primary, secondary, or cross memory lock) for a given processor have the same ASID, that address space is only processed once. If AMDPRDMP cannot determine the ASID of the cross memory address spaces or if they are not in the dump data set, the ASCB of the address space that was current when the dump was taken is used.
- PRINT JOBNAME — Obtain addresses of ASCBs associated with jobnames specified in the control statement.

Exit Module

MO11 Part 3

Extended Description	Module	Label
1 When AMDPRDMP encounters a PRINT control statement with the CURRENT or JOBNAME parameters, it gets the ASCB(s) associated with the parameter.		
If the exit control table (ECT) indicates the jobname/current exit is in effect, AMDPRDMP passes control to the user-exit module before beginning its processing.	AMDPRUIM	AMDUSRXT
• CURRENT — AMDPRDMP formats and prints the cross memory address spaces (home, primary, secondary, and cross memory lock). For SADMP cross memory address spaces, AMDPRDMP formats and prints ASCBs for each outline processor. For SVCDUMP or SYSMDUMP cross memory address spaces, if the cross memory address spaces cannot be determined, AMDPRDMP gets the address of the current ASCB and the current ASID from the CURASCB and QASID fields in COMMON respectively. If either value is zero, AMDPRDMP cannot print CURRENT data and issues a message to that effect.	AMDPRPCR	
Note that if the ASID of any of the four possible address spaces is the same, that address space is only printed once and a message is issued to that effect.		
• JOBNAME — AMDPRDMP builds a list of jobnames specified in the control statement. It compares the jobname specified in each ASCB with the names in the list. Whenever an ASCB is found with a specified jobname, a switch is set in the jobname list indicating the name has occurred.	AMDPRPJB	

PRDMP Diagram 10. The PRINT CURRENT/JOBNAME Control Statements (Part 3 of 4)

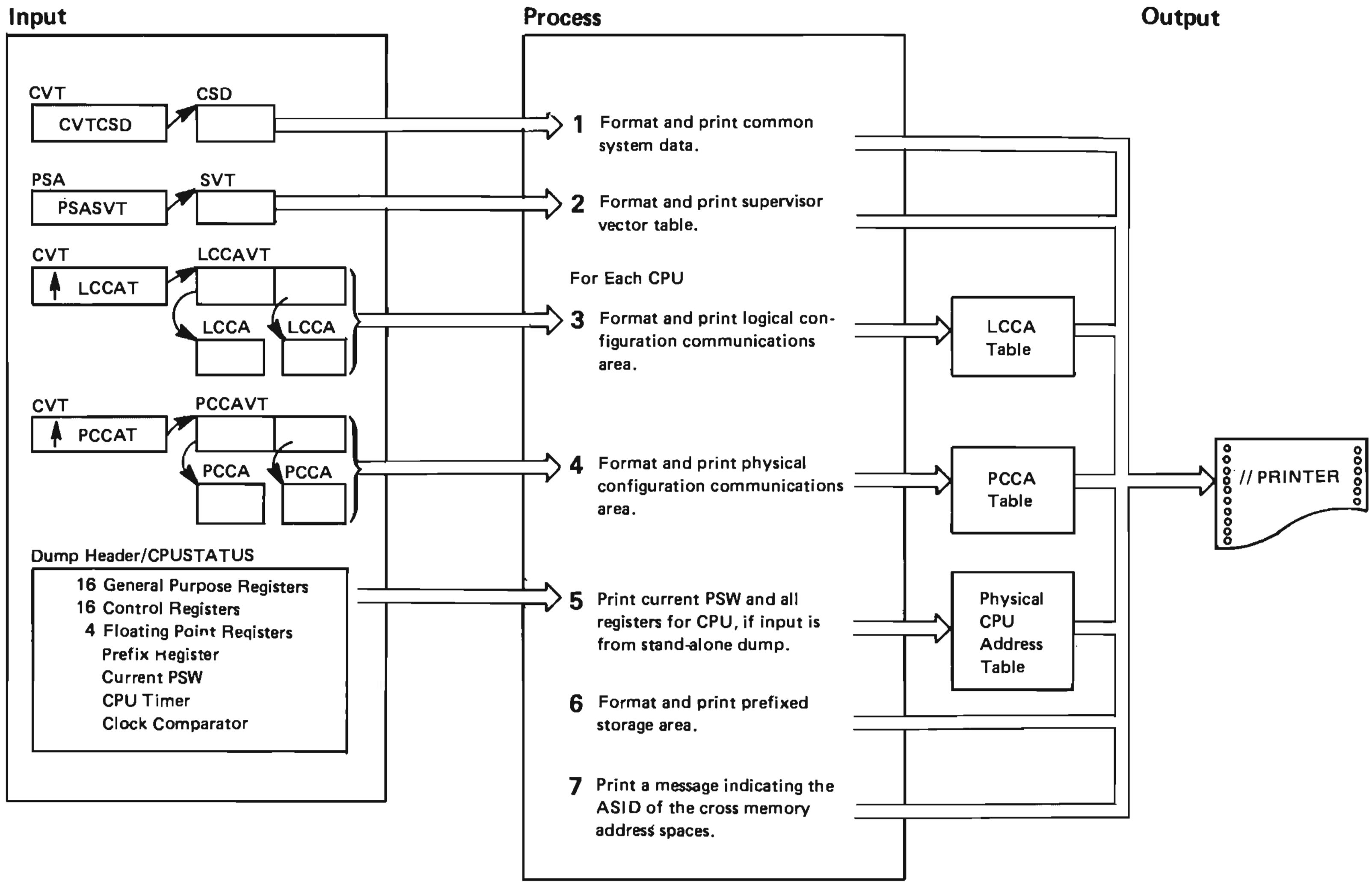


PRDMP Diagram 10. The PRINT CURRENT/JOBNAME Control Statements (Part 4 of 4)

Extended Description	Module	Label
2 AMDPRDMP uses the region descriptor to determine the beginning and ending addresses of the private area. Then it determines the unallocated space from the FBQEs for the ASCB. AMDPRDMP calculates addresses for space allocated to the region.	AMDPRDPS	PCBRTN
Then using the addresses in the parameter list, AMDPRDMP builds a PCB to contain the addresses.	AMDPRDPS	PCBENQ1
It enqueues the PCB in its proper sequence in the PCB queue, then reads and formats the storage areas specified in the PCBs. As the storage is written in the PRINTER data set, AMDPRDMP dequeues the PCBs.	AMDPRDPS AMDPRDPS	PRNTSTG
3 PRDMP invokes the SUMMARY verb exit with the FORMAT keyword for each ASID selected in Step 1. SUMMARY formats the ASCB and each TCB.	AMDPRUIM BLSQSUM1	AMDUSRXT
• RBs pointed to by the TCB.		
• Extended status blocks (XSBs) pointed to by each active request block (RB).		
• Queue of PCLINK stack elements (STKEs) associated with each XSB.		
• Load list.		
• Job pack queue area.		
• DEBs associated with the TCB.		
• Task input/output table (TIOT) for the TCB.		
• Data management control blocks for the TCB (DCB and IOB/ICB/LCB).	IECDAFMT	
• IOS control blocks for the TCB (XDBA (EXCPD) and UCB).	IECIOFMT	
• RTM control blocks for the TCB (RTCT, RTM2WA, EED, SCB, FRRS, and IHSA).	IEAVTFMT	
• XSBs and STKEs pointed to by the IHSA.	IEAVTFMT	

Extended Description	Module	Label
4 For the CURRENT parameter, AMDPRDMP writes the contents of the prefix register, the 16 general purpose registers, 16 control registers, the 4 floating point registers and the current PSW (if available) in the PRINTER data set. AMDPRDMP checks the GPRSFND bit in SWD of COMMON to determine	AMDPRDPS	PRTSTGO
whether it should write the registers. AMDPRDMP will obtain the current ASCB and ASID. The CVTTCBP points to a four word table. The fourth word of that table points to the current ASCB. If the current ASCB cannot be determined, the following comment is written:	AMDPRPCR	
'UNABLE TO ACCESS CURRENT TASK'		
For a SADMP, AMDPRDMP determines if the current address space is the dummy wait task.		
The address space is the dummy wait task when the PSAAOLD field in PSA points to the master scheduler's address space and the PSAANEW field points to WAIT's address space. The WAIT task has an ASID of zero. If the current ASCB is the dummy wait the following comment is printed:		
'CURRENT TASK is DUMMY WAIT TASK'		
5 AMDPRDMP prints the storage associated with the requested address spaces, and fills in PCBPL.	AMDPRDPS	AMDPRDPS
6 AMDPRDMP prints the TCB summary collected during formatting of control blocks.	AMDPRFSC	

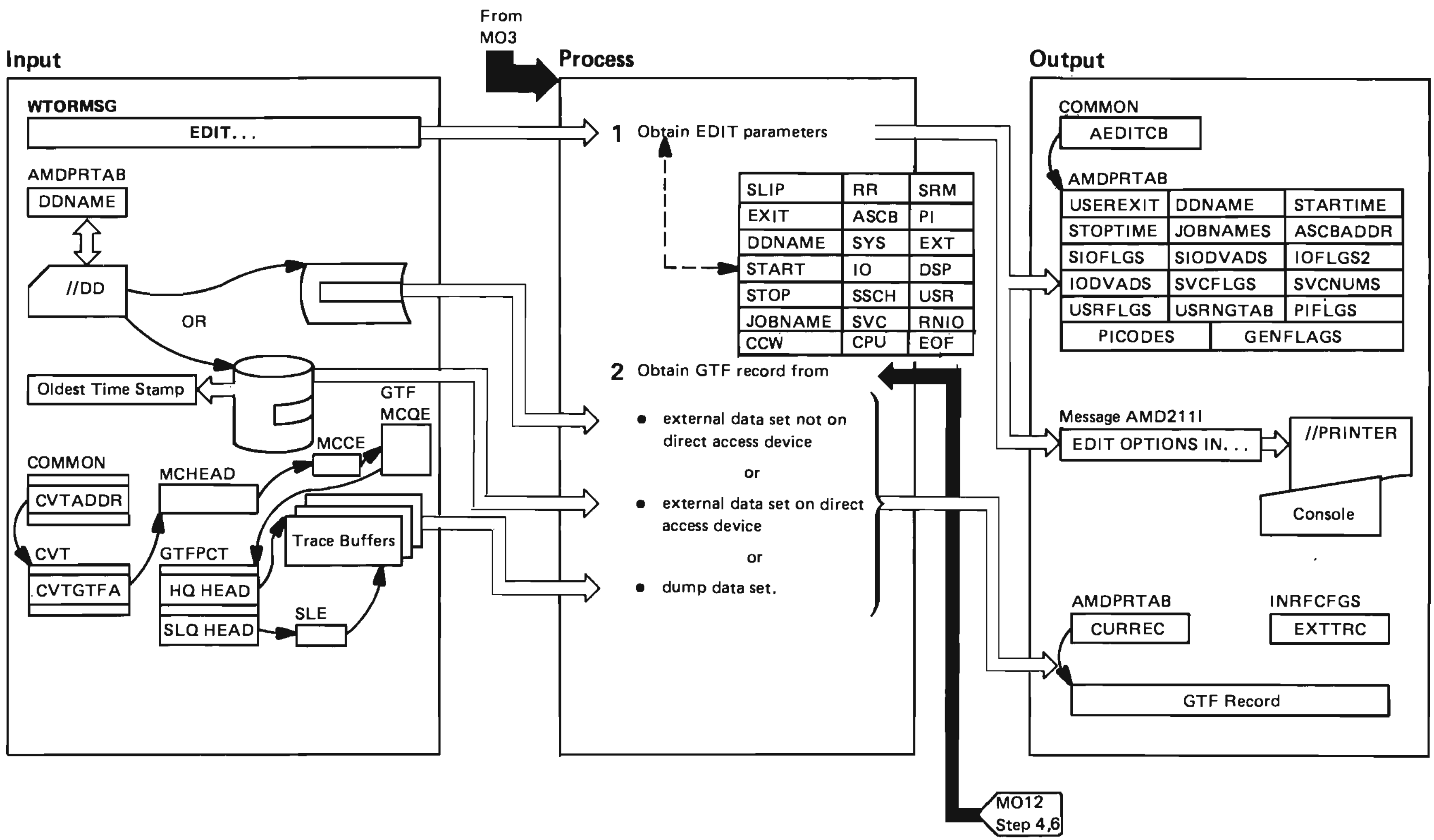
PRDMP Diagram 11. The CPUDATA Control Statement (Part 1 of 2)



PRDMP Diagram 11. The CPUDATA Control Statement (Part 2 of 2)

Extended Description	Module	Label
The CPUDATA control statement causes AMDPRDMP to format and print processor-related information.		
AMDPRGCD invokes AMDPRUIM to initialize the exit interface and then uses the Control Block Formatter exit service to format control blocks.		
1 AMDPRDMP uses the CVT pointer to locate, format, and print the common system data area (CSA).	AMDPRGCD	
2 AMDPRDMP uses the PSA pointer (PSASVT) to locate and print the supervisor vector table (SVT).	AMDPRGCD	
3 AMDPRDMP uses the CVT pointer to locate, format, and print the logical configuration communications area (LCCA) for a processor. If the LCCA vector table pointers are invalid, the prefixed storage area (PSA) pointer to the LCCA is used.	AMDPRGCD	
4 AMDPRDMP uses the CVT pointer to locate, format, and print the physical configuration communications area (PCCA) for a processor. If the PCCA vector table pointers are invalid, the program uses the pointer in the PSA to get the PCCA. Output comments are issued if pointers are invalid or if the control block cannot be found.	AMDPRGCD	
5 If the input data set is from AMDSADMP, the PSW and all registers for the processor are obtained from the CPUSTATUS record. Otherwise, AMDPRDMP gets this information from the dump header record. In either case, AMDPRDMP prints the current PSW and all registers for the processor.	AMDPRDPS	
A table of processor addresses is kept to avoid duplication of processing. An output comment is issued if the data cannot be printed.	AMDPRGCD	
6 AMDPRDMP formats and prints the prefixed save area (PSA) for the processor and the current PCLINK stack element (STKE) queue pointed to by the PSA.	AMDPRDPS	
7 AMDPRDMP prints a one line message that indicates the ASID of the cross memory address spaces (home, primary, secondary, and cross memory lock) held.	AMDPRGCD	

PRDMP Diagram 12. The EDIT Control Statement (Part 1 of 10)

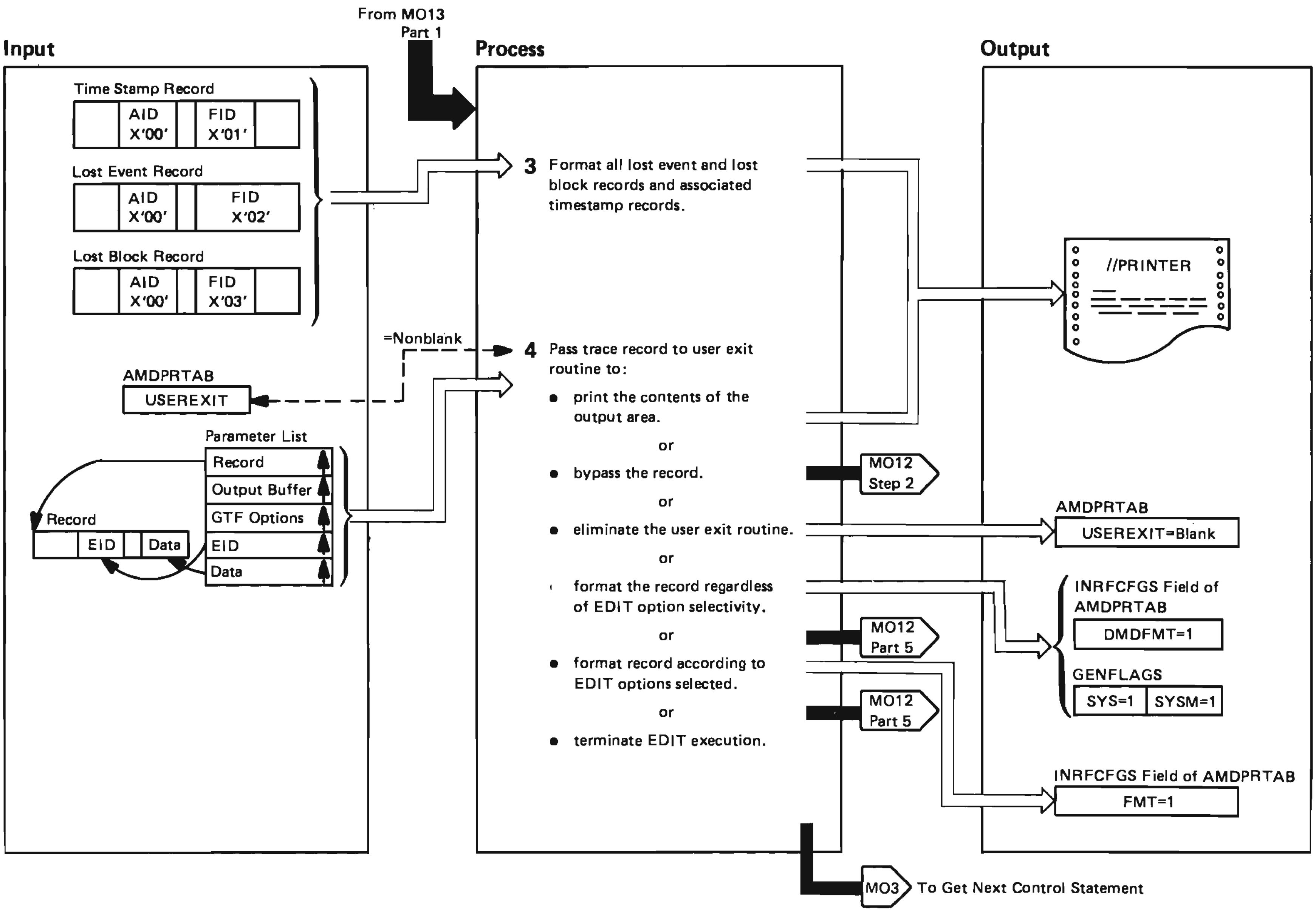


PRDMP Diagram 12. The EDIT Control Statement (Part 2 of 10)

Extended Description	Module	Label
1 When AMDPRDMP encounters an EDIT control statement, it initializes the EDIT communications table (AMDPRTAB) pointed to by the AEDITCB field of COMMON with the EDIT parameter values derived from the EDIT control statement. It resolves mutually exclusive parameter conflicts and sorts selected SSCH, IO, and USR values in ascending order. Then it issues a summary message (AMD211I) to inform the user of the EDIT options in effect. See *Service Aids* for a description of the EDIT parameters.	AMPRCTL AMDPROOT AMDPRSCN	
2 In order to obtain a GTF record for the first time, AMDPRDMP determines the source of the input data, either an external data set or a dump data set.	AMDPRGET	
• If a DDNAME is specified in the DDNAME field of AMDPRTAB, AMDPRDMP assumes that the associated DD statement in the AMDPRDMP JCL describes an external GTF data set. It sets on the EXTTRC flag in INRFCFGS field of AMDPRTAB, then checks the device to determine whether the external data set is on a direct access device. If not, it obtains the first record in the data set.		
• If the data set resides on a direct access device, AMDPRDMP positions to the record with the oldest timestamp. (If no timestamp is available, AMDPRDMP processes the data set as if it were not on a direct access device.)		

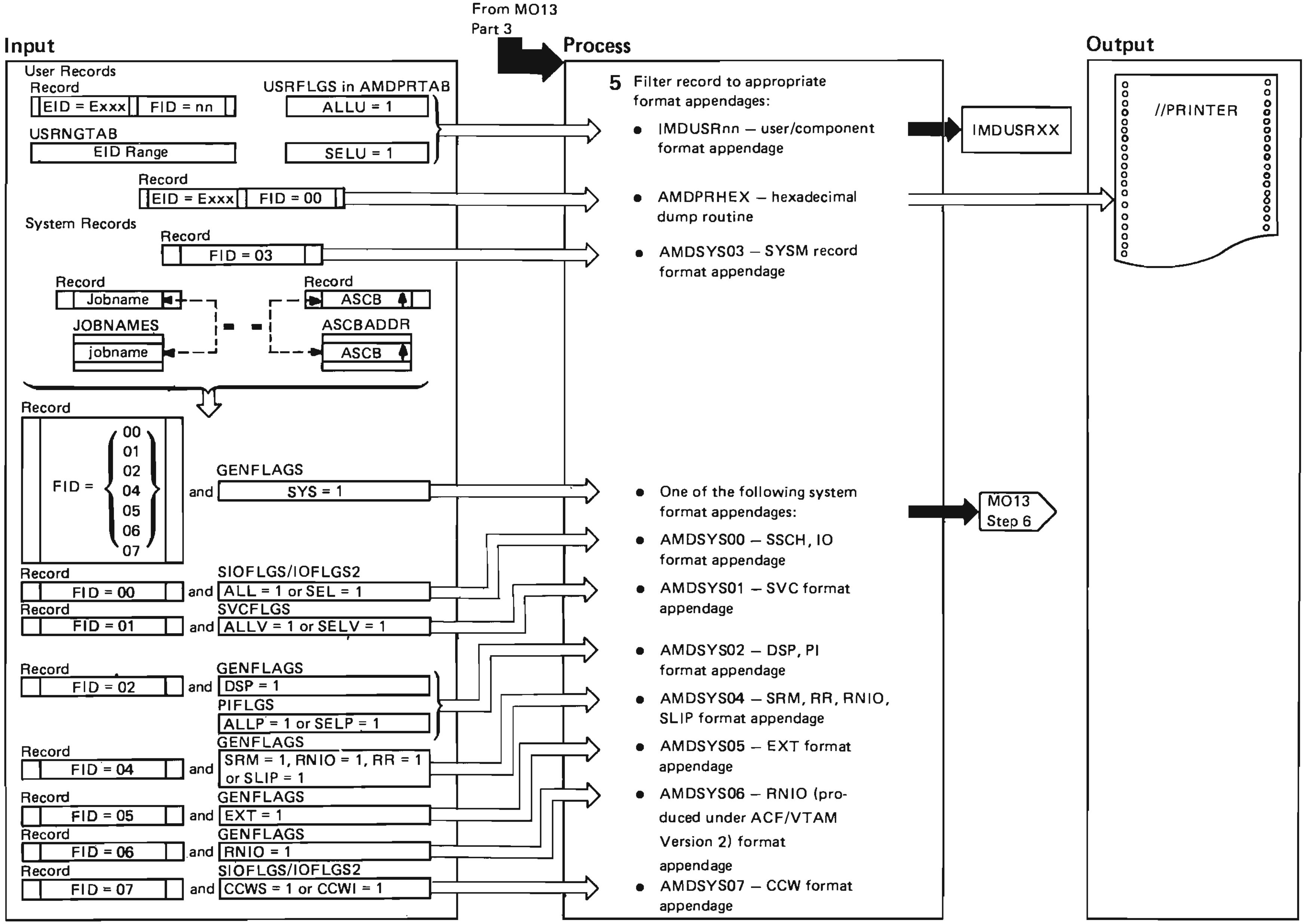
Extended Description	Module	Label
• If the GTF buffers are in a dump data set, AMDPRDMP obtains the address of the CVT in the dump from CVTADDR in COMMON. From the CVT, AMDPRDMP locates MCHEAD, the control table of the monitor call (MC) routing facility. From MCHEAD, AMDPRDMP locates the MC control element (MCCE) which in turn points to a chain of MC queue elements (MCQEs). AMDPRDMP scans the chain for the GTF application MCQE. The MCQE contains the address of the GTF control table, GTFPCT. If the dump is from AMDSADMP, AMDPRDMP uses the GTFPCT pointer (HQHEAD) to the most recent GTF buffers. AMDPRDMP also formats the GTFBLOKs in order from oldest to newest. If the dump is from SVC Dump, AMDPRDMP uses the GTFPCT pointer to the save queue (SLQ HEAD). The save list element chain is scanned for ASID of the dump. The buffers required are then located. (In either case, the queue pointer is to a buffer control block which points to the buffer.) AMDPRDMP stores the address of the record in the CURREC field of AMDPRTAB. (For a description of GTF output records, see the "Data Areas" section.)		

PRDMP Diagram 12. The EDIT Control Statement (Part 3 of 10)



Extended Description	Module	Label
3 If the record has an AID of X'00', it is a control record (timestamp, lost block, or lost data). AMDPRDMP initializes the output buffers using the FMTPTRN macro instruction and converts the record to the output format. The formatted records are written in the PRINTER output data set for later printing in the AMDPRDMP output listing.	AMDPRREC AMDPRCOM	
4 If the USEREXIT field of AMDPRTAB is non-blank, AMDPRDMP passes the record to the user exit module via a 5-word parameter list containing the address of a copy of the input record, the address of the output buffer, the address of the GTF option word, and the addresses of the EID and data fields in the record. On the return, the user exit routine instructs AMDPRDMP to do one of the following: • Print the output buffer. • Bypass the record and get the next one. • Eliminate the user exit (set the USEREXIT field to blanks). • Format the record regardless of the EDIT options in effect (set the SYSM and SYS bits on in GENFLAGS). • Format the record according to the EDIT options in effect. • Terminate a EDIT processing.	AMDPRFLT AMDPREXT	

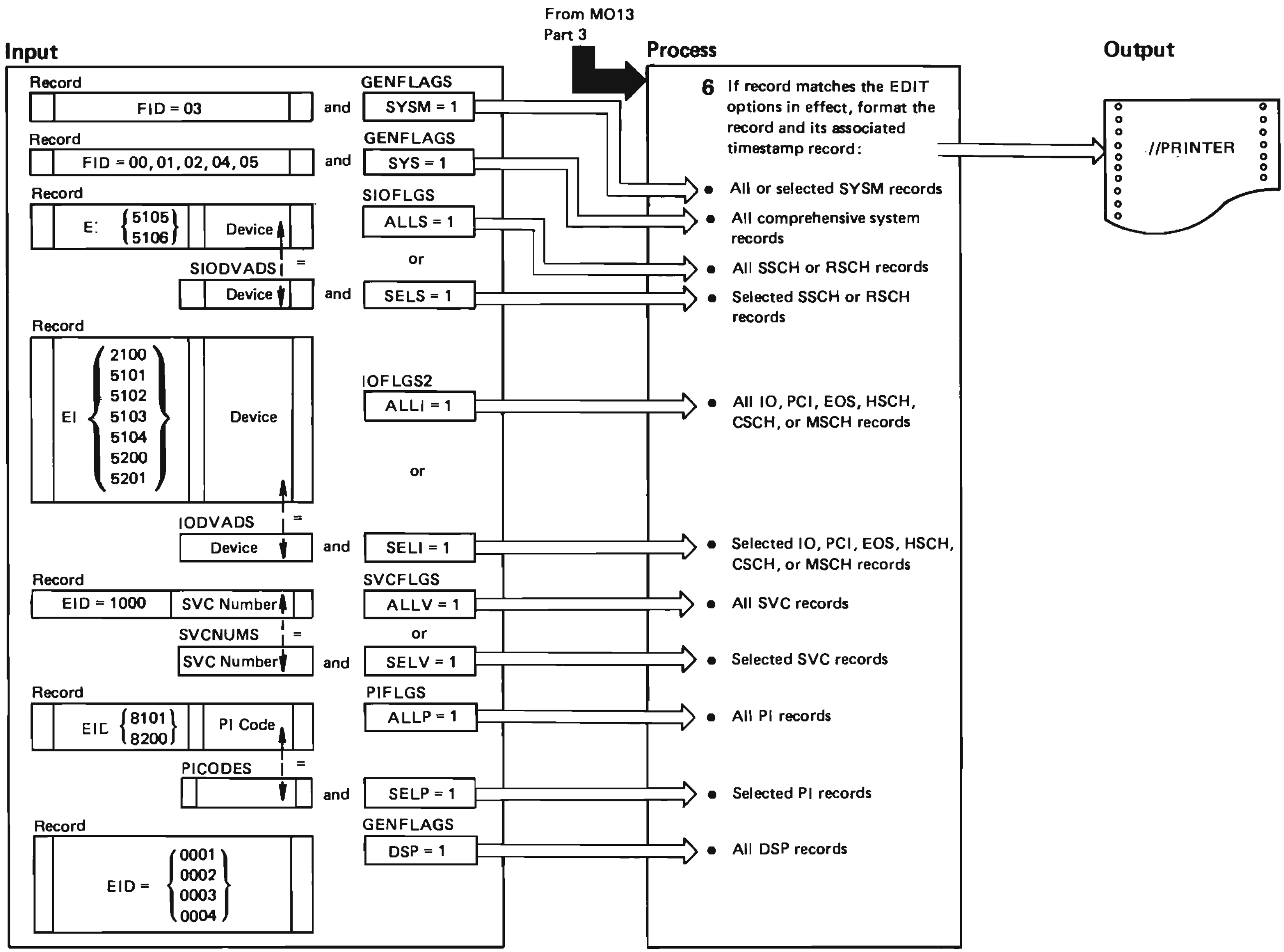
PRDMP Diagram 12. The EDIT Control Statement (Part 5 of 10)



PRDMP Diagram 12. The EDIT Control Statement (Part 6 of 10)

Extended Description	Module	Label
5 If the user exit specifies that the record should be formatted, or if no user exit is included, AMDPRDMP performs a variety of tests to filter the record to the appropriate format appendages:	AMDPRFLT	
• If the first character of the EID is E, the record is a user record. If the ALLU flag is on, or if the SELU flag is on and the EID in the record is included in the EID ranges specified in the USRNGTAB of AMDPRTAB, AMDPRDMP passes control to the user format appendage IMDUSRxx where xx is the FID in the record.	AMDPRAPP	
• If the FID in a user record is 00, the user wants the record to be dumped in hexadecimal, so AMDPRDMP passes the record to the hexadecimal dump routine, AMDPRHEX, located in AMDPRDMP.	AMDPRREC	
• If the FID of the record is 03, the record is a SYSM record. AMDPRDMP then uses the option bits in AMDPRTAB and the EID to determine whether the record is to be formatted. If so, AMDPRDMP passes control to the minimal record format appendage.	AMDPRAPP	
• If the record is not user or SYSM, AMDPRDMP interrogates the JOBNAMES field in AMDPRTAB. If jobnames have been specified, AMDPRDMP verifies that the jobname in the record matches one of the selected jobnames. If not, it performs a similar check for ASCBADDR and the ASCB address in the record. If no match at all occurs, the record is bypassed. If a match occurs, or if neither jobnames nor ASCB addresses were specified, AMDPRDMP filters the record to the proper system format appendage.	AMDPRFLT	
• If the SYS bit is on in the GENFLAGS field, AMDPRDMP passes control to AMDSYSxx where xx is the FID.	AMDPRAPP	

Extended Description	Module	Label
• If the SYS bit is off and the FID is 00, AMDPRDMP verifies that at least one bit is set on in the SIOFLGS or IOFLGS2 field, then passes control to the SSCH, IO comprehensive record format appendage.	AMDPRAPP	
• If the SYS bit is off and the FID is 01, AMDPRDMP verifies that at least one bit is on the SVCFLGS field, then passes control to the SVC comprehensive record format appendage.		
• If the SYS bit is off and the FID is 02, AMDPRDMP verifies that either the DSP flag in GENFLAGS is on or that one of the bits in the PIFLGS field is on, then passes control to the PI, DSP comprehensive record format appendage.	AMDPRAPP	
• If the SYS bit is off and the FID is 04, AMDPRDMP verifies that either the SRM, RNIO, RR, or SLIP flag in GENFLAGS is on, then passes control to the SRM, RNIO, RR, SLIP comprehensive record format appendage.	AMDPRAPP	
• If the SYS bit is off and the FID is 05, AMDPRDMP verifies that the EXT flag in GENFLAGS in on, then passes control to the EXT comprehensive record format appendage.	AMDPRAPP	
• If the SYS bit is off and the FID is 06, AMDPRDMP verifies that the RNIO flag in GENFLAGS is on and passes control to the format appendage routine (AMDSYS06).		
• If the SYS bit is off and the FID is 07, AMDPRDMP verifies that the CCWS bit in SIOFLGS or the CCWI bit in IOFLGS2 is on, then passes control to the CCW comprehensive record format appendage.	AMDPRDMP	

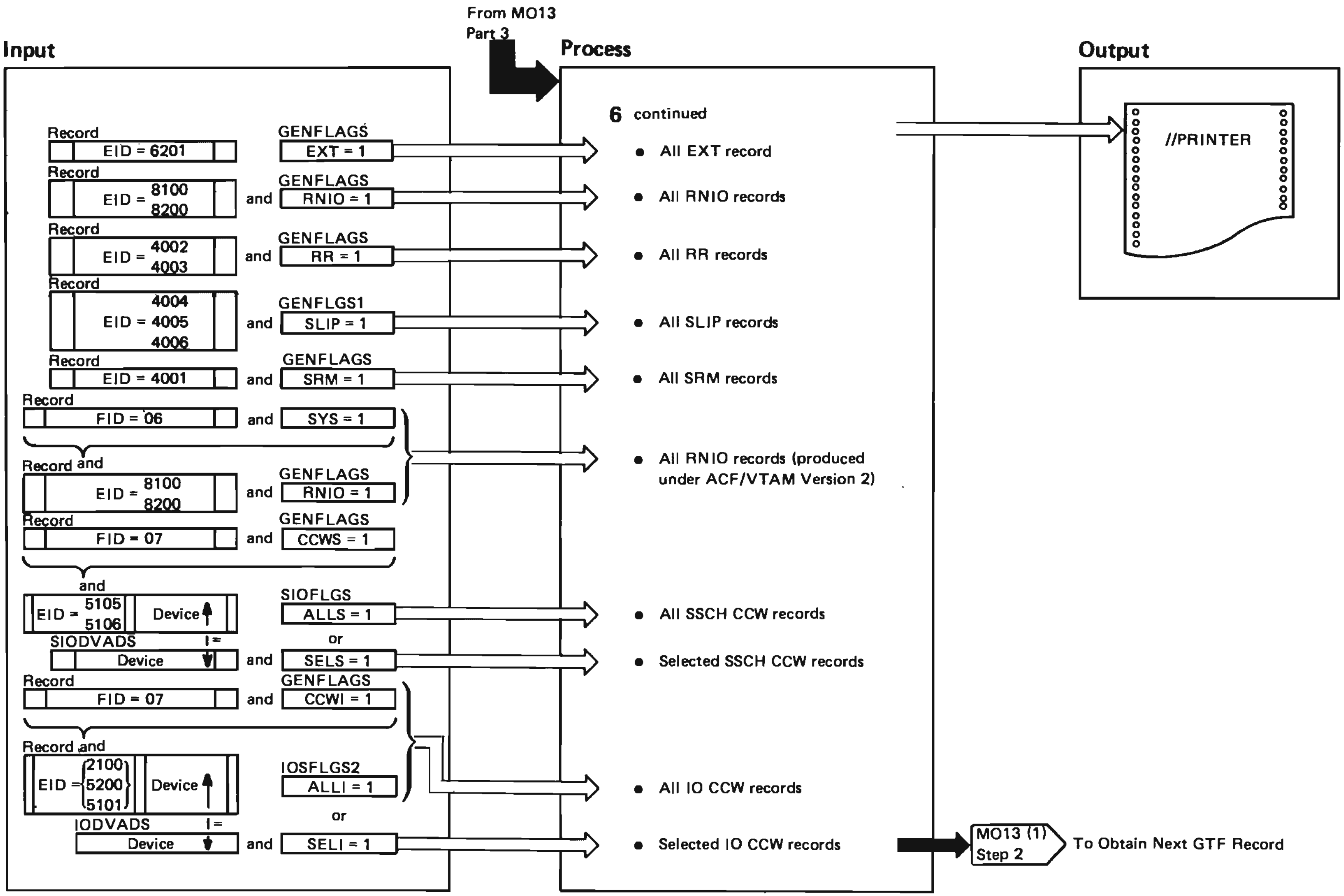

From MO13
Part 3
Input
Process
Output
Record
FID = 03
and
GENFLAGS
SYSM = 1
Record
FID = 00, 01, 02, 04, 05
and
GENFLAGS
SYS = 1
Record
E: 5105 5106
Device
SIOFLGS
ALLS = 1
SIODVADS
=
or
Device
and
SELS = 1
Record
EI 2100 5101 5102 5103 5104 5200 5201
Device
IOFLGS2
ALLI = 1
or
IODVADS
=
Device
and
SELI = 1
Record
EID = 1000
SVC Number
SVCFLGS
ALLV = 1
SVCNUMS
=
or
SVC Number
and
SELV = 1
Record
EID 8101 8200
PI Code
PIFLGS
ALLP = 1
PICODES
=
and
SELP = 1
Record
EID = 0001 0002 0003 0004
GENFLAGS
DSP = 1
6 If record matches the EDIT options in effect, format the record and its associated timestamp record:
• All or selected SYSM records
• All comprehensive system records
• All SSCH or RSCH records
• Selected SSCH or RSCH records
• All IO, PCI, EOS, HSCH, CSCH, or MSCH records
• Selected IO, PCI, EOS, HSCH, CSCH, or MSCH records
• All SVC records
• Selected SVC records
• All PI records
• Selected PI records
• All DSP records
//PRINTER

PRDMP Diagram 12. The EDIT Control Statement (Part 8 of 10)

Extended Description	Module	Label
6 The system format appendages perform more tests on the record to ensure that the record qualifies for formating according to the EDIT options that are in effect.		
• AMDSYS03 determines whether all SYSM records are to be formatted (SYSM bit in GENFLAGS is on) or just selected event records.	AMDSYS03	
• AMDSYS00, AMDSYS01, AMDSYS02, AMDSYS04, AMDSYS05, and AMDSYS06 check the SYS bit in GENFLAGS to determine whether all comprehensive records should be formatted or just selected event records.	AMDSYS00 AMDSYS01 AMDSYS02 AMDSYS04 AMDSYS05 AMDSYS06	
• If the user specifies the CPU option, AMDSYS00 verifies that the given CPU value matches the CPU ID in the IO and/or SSCH record.	AMDSYS00	
• AMDSYS00 determines whether the record is on IO or SSCH record by checking the EID field. If it is an SSCH record, AMDSYS00 checks the SIOFLGS field. If the ALLS bit in on, or if the SELS bit is on and the device address in the record is among those specified in SIODVADS, AMDSYS00 formats the record. If the record is an IO record, AMDSYS00 checks the IOFLGS2 field. If the ALLI bit is on, or if the SELI bit is on and the device address in the record is among those included in IODVADS, AMDSYS00 formats the IO record. Either the SSCH or the SIO option cause formatting of SSCH and RSCH records. The IO option causes formatting of the IO, PCI, EOS, HSCH, CSCH, and MSCH records.	AMDSYS00	

Extended Description	Module	Label
• AMDSYS01 checks the SVCFLGS field. If the ALLV bit is on, or if the SELV bit is on and the SVC number of the record corresponds to one in the SVCNUMS field, AMDSYS01 formats the record.	AMDSYS01	
• AMDSYS02 checks the record EID to determine the record type. If the record is a PI record, it checks the PIFLGS field. If the ALLP bit is on, or if the SELP bit is on and the PI code in the record corresponds to one in the PICODES field, AMDSYS02 formats the record. If the record is a DSP record, AMDSYS02 verifies that the DSP bit in GENFLAGS is on, then formats the record.	AMDSYS02	

PRDMP Diagram 12. The EDIT Control Statement (Part 9 of 10)



Extended Description	Module	Label
6 continued		
• AMDSYS04 checks the record EID to determine the record type. If the record is an SRM record, it verifies that the SRM bit in GENFLAGS is on, then formats the record. If the record is an RR record, AMDSYS04 verifies that the RR flag in GENFLAGS is on, then formats the record. If the record is an RNIO record, AMDSYS04 verifies that the RNIO flag in GENFLAGS is on and then formats the record. If the record is a SLIP record, AMDSYS04 verifies that the SLIP flag in GENFLGS1 is on and then formats the record.	AMDSYS04	
• AMDSYS05 verifies that the EXT flag in GENFLAGS is on, then formats the record.	AMDSYS05	
• AMDSYS06 verifies that the RNIO flag in GENFLAGS is on, then formats the record.	AMDSYS06	
• If the user specifies the CPU option, AMDSYS07 verifies that the given CPU value matches the CPU ID in the IO and/or SSCH record.	AMDSYS07	
• AMDSYS07 checks the CCWS bit and the CCWI bit in GENFLAGS to determine whether CCW trace records should be formatted for SSCH and IO events. If either or both CCW bits are on, AMDSYS07 determines whether the record is an IO or SSCH record by checking the EID field. When the CCWS bit is on, and the record is an SSCH record, AMDSYS07 checks the SIOFLGS field. If the ALLS bit is on, AMDSYS07 formats all SSCH CCW records. If the SELS bit is on and the device address in the record is among those specified in SIODVADS, AMDSYS07 formats selected SSCH CCW records. When the CCWI bit is on, and the record is an IO record, AMDSYS07 checks the IOFLGS2 field. If ALLI bit is on, AMDSYS07 formats all IO CCW records. If the SELI bit is on and the device address in the record is among those specified in IODVADS, AMDSYS07 formats selected IO CCW records.	AMDSYS07	
After the record is formatted, AMDPRDMP writes it in the PRINTER output data set for later printing: AMDRPDMP then obtains the next GTF record for formatting.	AMDPRCOM	

Section 3: Program Organization

This section describes how AMDPRDMP is organized to carry out the formatting and printing of information. It has three parts:

- A description of program loading.
- Module calling sequences for AMDPRDMP initialization, control statement processing and control statement execution.
- A description of the data-reading operation.

Program Loading

As shown in Figure 3-2, there are eight modules that are resident in virtual storage and link-edited into the single AMDPRDMP load module during system generation:

- AMDPRCTL is the main control module. It initializes the AMDPRDMP program and the common communication area, scans the user control statements, and initiates the execution of the functions requested by the control statements.
- AMDPRCOM contains the common communication area (COMMON) used by all AMDPRDMP modules; it also contains a number of small service routines used by other AMDPRDMP modules.
- AMDPRRDC is the read control module.
- AMDPRSEG is the segment loading module. It loads AMDPRDMP modules from JOBLIB, STEPLIB, or SYS1.LINKLIB into virtual storage when they are needed to perform functions requested by format control statements.
- AMDPRPMG is a message module that contains the addresses and text of AMDPRDMP messages.
- AMDPRUIM is the exit interface module. It loads the service modules provided for the exits and invokes BLSQECT to link to exits.
- AMDPREAD reads the dump records and builds maps of the dump data set if SYSUT1 is not defined.
- AMDPRGSA contains glue service routines for user exit storage access, and print and index services.

In addition to the modules that are resident in virtual storage, there are four other categories of AMDPRDMP modules that are resident in SYS1.LINKLIB:

- Executor modules for all format control statements except EDIT.
- Service modules for all format control statements except EDIT.
- EDIT modules.
- Read initialization modules.

Executor Modules

AMDPRGCD	AMDPRNUC	AMDPRPJB
AMDPRLPA	AMDPRPCR	AMDPRPMS

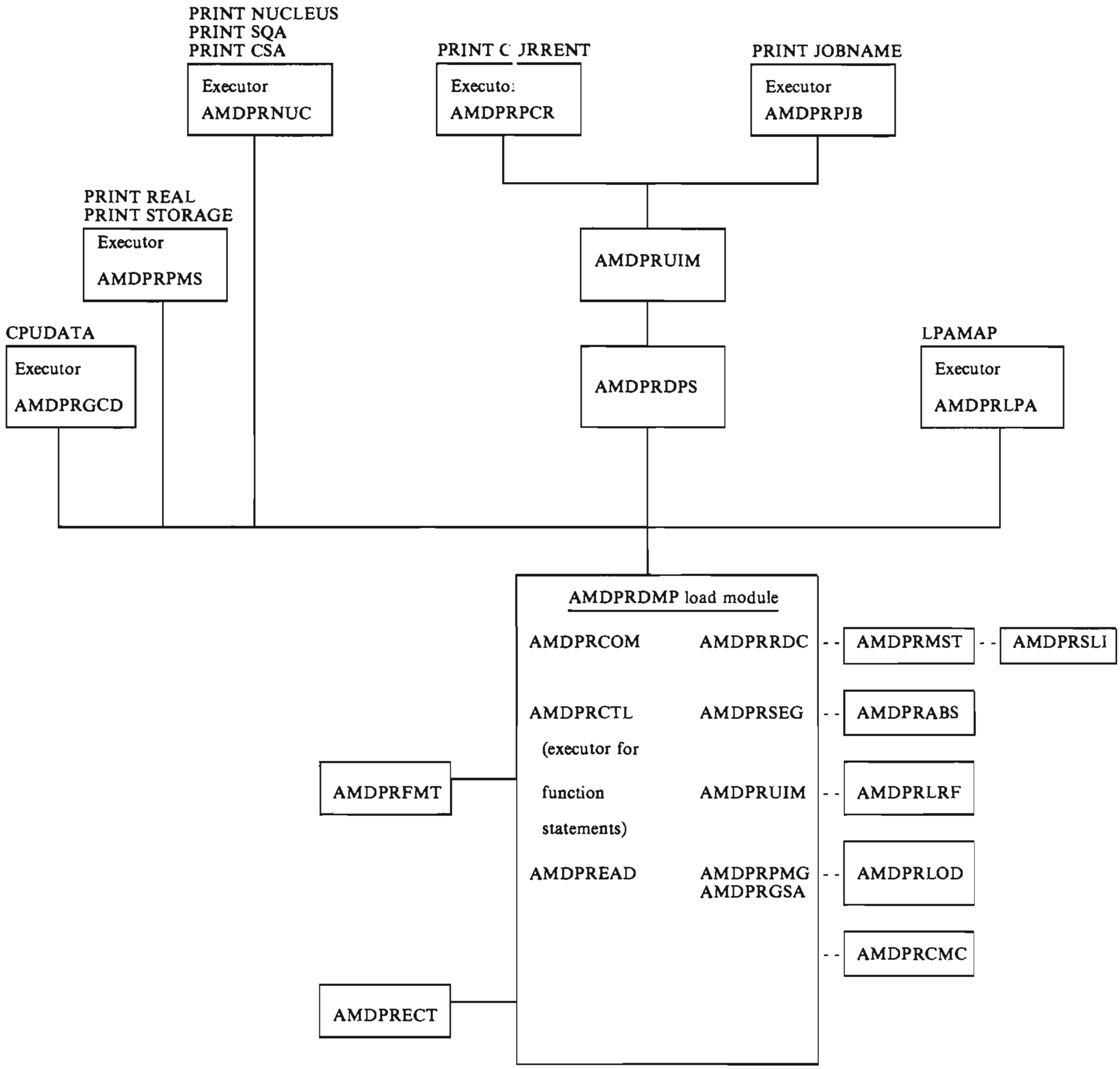


Figure 3-2. AMDPRDMP Loading

Executor modules govern the execution of functions requested by all AMDPRDMP control statements except EDIT. The EDIT modules form a category of their own.) The executor routines for function control statements (CVT, TITLE, GO, ONGO, NEWDUMP, NEWTAPE, SEGTAB, and END) are part of AMDPRCTL. The executor modules for SUMMARY, FORMAT, LPAMAP, CPUDATA, and PRINT are in SYS1.LINKLIB. They are brought into virtual storage by AMDPRSEG just before the associated format control statement is executed. All other control statements are user exits whose program modules are defined in the ECT. They are not loaded by AMDPRSEG, but are linked to AMDPRUIM via BLSQECT.

Service Modules

AMDPRDPS	AMDPRFMT
AMDPRECT	AMDPRFUB

Service modules perform functions that are either common to several executor modules or can be used by user-exit modules. Some small service routines are contained in AMDPRCOM. However, the larger service modules are in SYS1.LINKLIB. Like the executor modules for these control statements, the service modules in SYS1.LINKLIB are brought into virtual storage by AMDPRSEG just before the control statement is executed. The service modules for user-exit routines are loaded by AMDPRUIM during initialization and are resident for the remainder of AMDPRDMP processing.

EDIT Modules

AMDPRAPP	AMDPRFRM	AMDPRSMG	AMDSYS02	AMDSYS07
AMDPREID	AMDPRGET	AMDPRSN2	AMDSYS03	AMDSY101
AMDPREXT	AMDPROOT	AMDPRSN3	AMDSYS04	IMDUSRF9
AMDPRFLT	AMDPRREC	AMDSYS00	AMDSYS05	IMDUSRFE
AMDPRFMG	AMDPRSCN	AMDSYS01	AMDSYS06	IMDUSRFF

EDIT modules are loaded into virtual storage from SYS1.LPALIB to execute the function requested by the EDIT control statement. The EDIT function executes in two major loads: EDIT keyword processing and input data set formatting (editing). Figure 3-3 represents maps of the AMDPRDMP storage area during the two loads. Storage structure during the second load depends on the type of input data set, dump or external.

The three parts of Figure 3-3 contain the resident modules for AMDPRDMP (described previously). Figure 3-3A also contains the EDIT initialization module (AMDPROOT) and the keyword scan modules (load module name AMDPRSCN). Load module AMDPRSCN contains the following EDIT modules:

- AMDPRSCN, the EDIT keyword isolation module.
- AMDPRSCN2, the EDIT keyword processing module.
- AMDPRSCN3, the EDIT control statement termination module.
- AMDPREID, the EID table.
- AMDPRSMG, a message module.

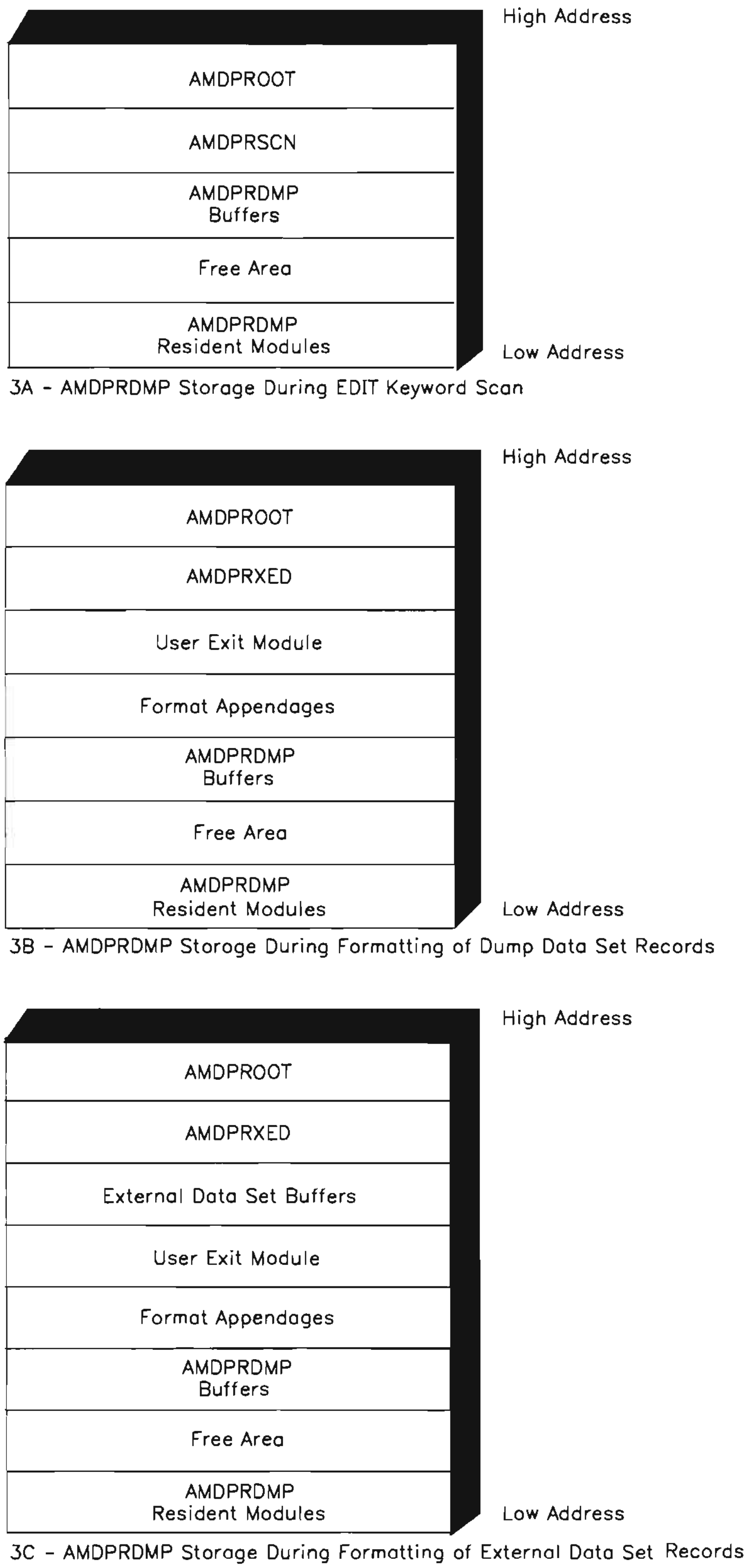


Figure 3-3. AMDPRDMP Storage During EDIT Control Statement Execution

The remainder of AMDPRDMP's storage area is buffers used by other AMDPRDMP functions. If EDIT is not the first control statement for this execution of AMDPRDMP, module AMDPREAD is also in storage.

Figure 3-3B depicts AMDPRDMP's storage area after EDIT keyword scan, if GTF buffers are to be edited from a dump data set. AMDPROOT remains in storage because it contains the EDIT communication table (AMDPRTAB) needed by all EDIT modules. Load module AMDPRXED has replaced AMDPRSCN in storage. AMDPRXED contains the following EDIT modules:

- AMDPRFRM, the EDIT execution control module.
- AMDPRGET, the EDIT I/O module.
- AMDPRFLT, the EDIT record filter module.
- AMDPREXT, the user exit interface module.
- AMDPRAPP, the format appendage interface module.
- AMDPRREC, the control record format and hexadecimal dump module.
- AMDPRFMG, the EDIT message module.

Figure 3-3B also contains system and user/component format appendages that are loaded by AMDPRAPP, using AMDPRSEG, to format input records. The following are format appendages:

- AMDSYS00, AMDSYS01, AMDSY101, AMDSYS02, AMDSYS03, AMDSYS04, AMDSYS05, AMDSYS06, AMDSYS07 — system format appendages.

- IMDUSRF9 — VSAM format appendage.

- IMDUSRFE — SAM/PAM/DAM format appendage.

- IMDUSRFF — OPEN/CLOSE/EOV component format appendage.

The remainder of the AMDPRDMP storage area is occupied by AMDPRDMP buffers that are needed to obtain information from the dump data set and optional user exit modules.

Figure 3-3C differs from 3B in that external data set buffers, into which records from an external data set are read, occupy part of the storage devoted to format appendages in 3B. Following these buffers are the format appendages, and any remaining storage is occupied by AMDPRDMP buffers left from a different AMDPRDMP operation, and module AMDPREAD.

Read Initialization Modules

AMDPRLOD	AMDPRABS
AMDPRMST	AMDPRLRF
AMDPRSLI	AMDPRCMC

These modules are brought into virtual storage from SYS1.LINKLIB as they are needed. The functions of the modules are as follows:

- AMDPRLOD, the work data set load module, loads the input data set into SYSUT1 or SYSUT2.

- AMDPRMST is invoked to store information from the input data set in the common communication area (COMMON). The information is used during the execution of format control statements.
- AMDPRSLI obtains dump data information describing the boundaries and layout of the storage map, and saves this information in the print dump COMMON area. AMDPRSLI is called by AMDPRMST, by the executors of the PRINT NUCLEUS, SQA, CSA, and STORAGE control statements and by AMDPRFUB (AMDPRFUB determines the private boundaries of a given address space).
- AMDPRABS formats and prints the dump abstract title page.
- AMDPRLRF formats and prints the console-initiated loop trace records.
- AMDPRCMC determines cross memory information for the dumped system and saves it in COMMON.

Reading Input Data

AMDPRDMP maintains a chain of buffers to contain input dump data. The resident read control module, AMDPRRDC, manages the buffer chain and handles requests for dump data from the other modules. When AMDPRRDC receives a request for data not in a buffer, the module calls AMDPREAD to read the data if possible. Following are descriptions of the way these modules perform data reading and of the manipulation of input buffers.

Read Control Module — AMDPRRDC

AMDPRRDC receives control from the other AMDPRDMP modules via a BRREAD macro instruction (see Section 6). When the DATA function is indicated by the macro instruction, it is a request for input dump data. AMDPRRDC's handling of the request depends on whether data requested is for a CPU ID or for a real or virtual address within the dump.

CPU Data Request: AMDPRRDC returns a pointer to the data which is set up in a fixed format. (See Section 5: Data Areas for the format of the CPU Status Area.) AMDPRRDC gets the information from COMMON; however, if input is from AMDSADMP, AMDPRRDC gets the information by searching the AMDPRDMP buffers for the CPU status record for that CPU ID. If the record is not in a buffer, AMDPRRDC calls AMDPREAD to read the record. If a read is unsuccessful AMDPRRDC branches to the error handler in effect.

Note: The dump header record is saved during print dump initialization processing. A pointer to this data is saved in COMMON and is available to all print dump modules. Requests for the header record do not use the BRREAD facility. They directly access the information based on the address initialized in COMMON.

Requested Data at Real Address: AMDPRRDC verifies that the input contains real storage data and that the address (rounded down to a 4K boundary) is valid. Then AMDPRRDC determines whether the requested data is already in a buffer and, if so, returns a pointer to the buffer. If the data cannot be located in a buffer, AMDPRRDC calls AMDPREAD to read the data. If an I/O error occurs or if any of the validity checks fails, or if the data is not in the input dump, AMDPRRDC branches to the error handler in effect.

Requested Data at Virtual Address: AMDPRRDC rounds the requested address down to a 4K boundary and performs address prefixing, if necessary, using the PREFXRBV value in COMMON. Then AMDPRRDC attempts to locate the data with one of the following results.

- The data is already in a buffer. AMDPRRDC returns a pointer to the data unless the buffer is marked for invalid data. In that case the active error handler receives control.

- The data is not in a buffer and the dump contains data from real storage equivalent so it can get the data from the real portion of the dump. If the data is not available in the real portion, AMDPRRDC attempts to get it from the virtual dump, if input contains virtual storage data. In any case, AMDPRRDC searches the input buffers for the data and calls AMDPREAD to read the data if it is not in a buffer. Finally, AMDPRRDC returns a pointer to the data in a buffer, or it branches to the active error handler if the buffer is marked as having invalid data.

- The data is not in a buffer and the dump contains only virtual data. AMDPRRDC calls AMDPREAD to read the data from the virtual address. AMDPRRDC checks the validity of the returned data and either returns to the caller or branches to the active error handler, if the data is marked invalid.

Length Specified for Requested Data: AMDPRRDC adds the requested data address to the requested length (up to 4K). If the total crosses a page boundary, AMDPRRDC does the following:

- Obtains a 4K buffer to store the requested data.

- Calls AMDPREAD to obtain the page containing the requested address.

- Moves the range from the first page to the beginning of the buffer.

- Calls AMDPREAD to obtain the page containing the remainder of the range.

- Moves the remainder of the range to the buffer.

- Returns the address of the 4K buffer.

- If the requested address range does not cross a page boundary, AMDPRRDC processes the request normally.

Read Buffer Manipulation

To minimize the number of I/O operations required to obtain information from the input data set without constraining the available storage, AMDPRRDC obtains a fixed number of dump data set buffers. In addition, it assigns a priority to the buffers so that frequently-used information remains in storage where it can be accessed without an I/O operation.

AMDPRDMP builds a buffer map for each buffer, then chains the buffer maps together in a queue. The highest priority buffer (represented by the first buffer map in the queue) contains the storage block that was last referenced. When a different storage block is required, AMDPRRDC rechains the buffer maps to cause the buffer containing the required block to be represented first in the buffer map queue. In this way, frequently-used data is represented first in the buffer map queue, and data that is not referenced is pushed down in the queue until it eventually is replaced by new dump information.

Rechaining the Buffers: As AMDPRRDC fills requests for data, the buffers are rechained via their maps as follows (see also Figure 3-4):

- If there is a request for data that is not in a buffer, and there is an empty buffer available (call it buffer D as in PRDMP Figure 3-4), AMDPRRDC calls AMDPREAD to read the data into buffer D, enqueues the buffer D map as the first buffer map (most-recently referenced buffer), then chains the buffer D map to point to the buffer map for the buffer that was previously most-recently referenced (buffer C in Figure 3-4).

- If there is a request for data that is not in a buffer, and there re no empty buffers available, AMDPRRDC releases the buffer whose map is last in the queue (buffer A in Figure 3-4). It places the address of buffer A map in the buffer map queue base, chains the buffer A map to point to the buffer map previously first in the queue (buffer D in Figure 3-4), removes the buffer A map from its former place in the queue, and calls AMDPREAD to read the data into buffer A.

- If there is a request for data that is already in a buffer (buffer B in Figure 3-4), AMDPRRDC places the address of the buffer B map first in the queue, chains the buffer B map to point to the buffer map that was previously first in the queue (buffer D in Figure 3-4), and removes the buffer B map from its former place in the queue.

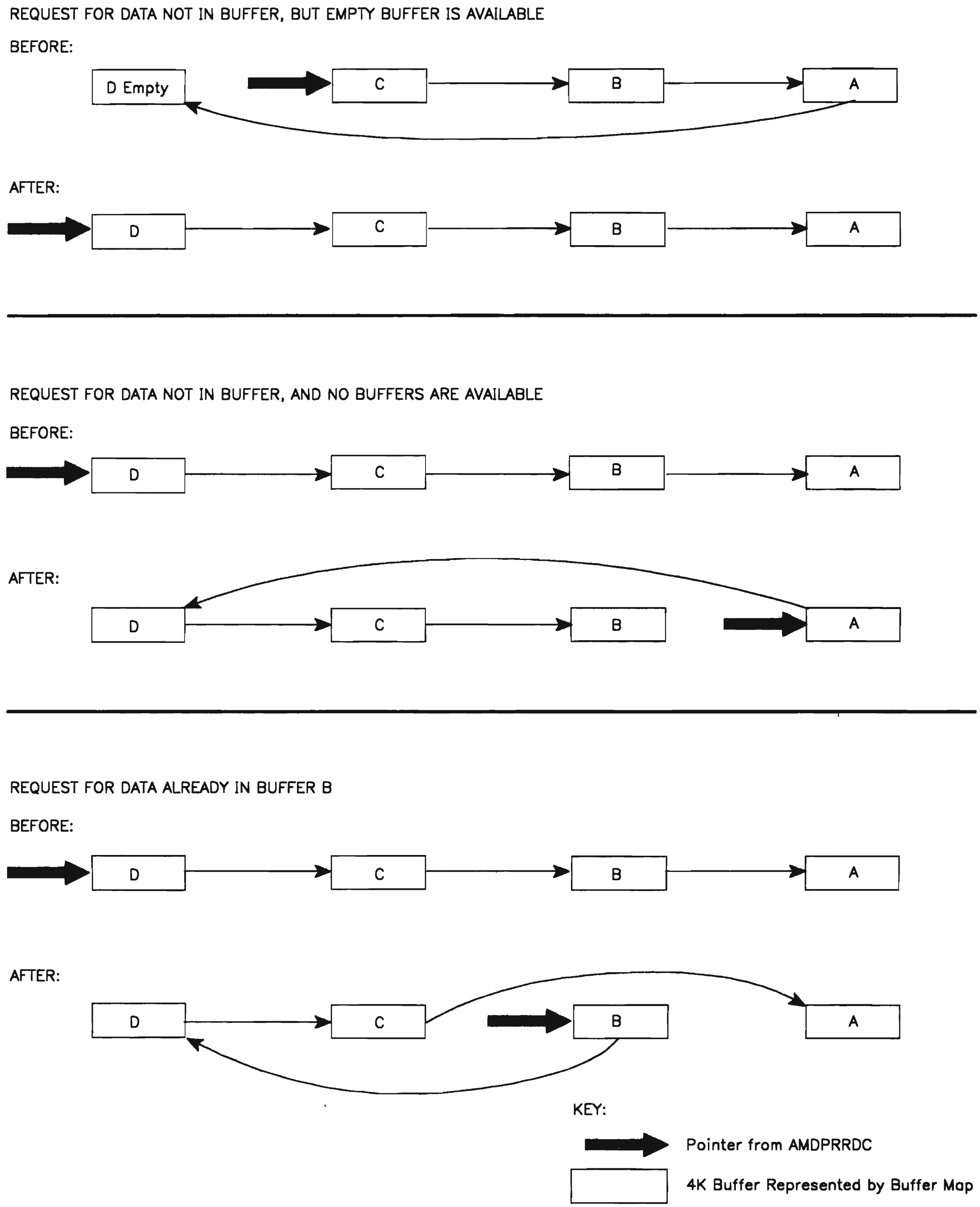


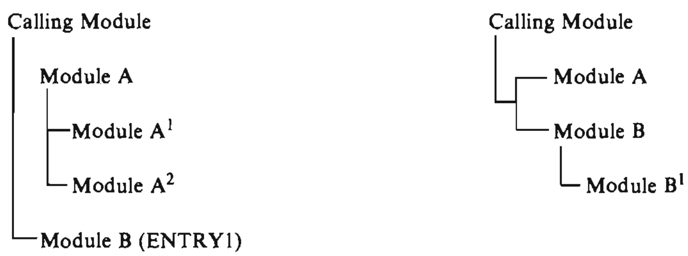
Figure 3-4. Read Buffer Chaining

Module Calling Sequences for AMDPRDMP Functions

The following maps show sequential flow of AMDPRDMP modules. Each map indicates the active modules for a specific function and describes the operations performed by those modules. Entry point names are also provided where they may differ from the module names. PRDMP Figure 3-5 explains the design of sequence map.

For more detailed descriptions of the functions, see the diagrams in Section 2: Method of Operation. A reference to the corresponding diagram accompanies each map.

Single Path Flow | **Alternate Path Flow**



The calling module first calls A, which then calls A^1 and A^2. When A returns, the caller passes control to B at entry point ENTRY1.

The calling module determines whether to call A or B for this operation. If B receives control, it calls B^1 before returning to the caller.

Figure 3-5. Example of Calling Sequence Map

Initialization (Diagram 2)

- AMDPRCTL (AMDPRCTL) (AMDPRMSC) — Analyzes EXEC statement parameters and stores value s in COMMON; reads in control statements. Scans first formatting statement.
  - AMDPRCOM (AMDPRWTR, AMDPRMSG) — Opens PRINTER and/or INDEX data sets. Gets title from the operator.
  - AMDPRUIM (AMDPRUIM) — Initializes exit interfaces: Puts ABDPL and SRA addresses in common.
  - AMDPRSEG — (after the first formatting statement has been read)
    - AMDPRRDC (AMDPRRDC) — INIT function: Loads AMDPRLOD (if input on tape), initializes and maps buffers, loads and deletes AMDPRMST, and builds ASCBMAP.
      - AMDPRLOD — If tape input, loads dump to SYSUT1 workfile and builds dump maps.
      - AMDPREAD — If SYSUT1 was not defined, builds dump maps.
      - AMDPRMST — Initializes COMMON with dump data including symptoms extracted from the dump header record for an SVC dump to be presented as part of the title page output produced by this module.
        - AMDPRSLI — Provides information related to the boundaries describing the various areas that comprise the storage layout.
        - AMDPRUIM — Invokes BLSQECT to link to the user-header exit modules
        - AMDPRRDC (AMDPRRDD) — Locates the dump records
          - AMDPREAD — Reads records into buffers.
      - AMDPRCMC — Determines cross memory ASIDs.
      - AMDPRABS — Prints the abstract title page.
      - AMDPRLRF — Formats and prints the instruction trace data (created by the console-initiated loop recording).

Function Statement Processing (Diagram 4)

- AMDPRCTL (AMDPRMSC) — Processes CVT, GO, ONGO, NEWDUMP, NEWTAPE, SEGTAB, and TITLE statements.
  - AMDPRCOM (AMDPRSYN) — Writes syntax error messages.

LPAMAP (Diagram 4)

- AMDPRCTL (AMDPRMSC) — Scans control statements.
  - AMDPRLPA — Executor: controls formatting and printing of LPA map.
    - AMDPRCOM (AMDPRGFR, AMDPRWTR, AMDPRMSG) — Writes LPA title, formats output lines, writes messages.
    - AMDPRRDC (AMDPRRDD) — DATA function: locates LPA records.
      - AMDPREAD — Reads records into buffers.

END Statement (Diagram 5)

- AMDPRCTL (AMDPREND) — Prints execution statistics or causes transfer of dump to SYSUT2 (if END is the only statement).
  - AMDPRRDC (AMDPRRDC) — INIT function: loads AMDPRLOD.
    - AMDPRLOD — Transfers dump data set to temporary workfile.
  - AMDPRCOM (AMDPRMSG) — Prints statistic messages and prints the dump index on the PRINTER file (if the INDEX DD is not included in the JCL).
- AMDPRCTL (AMDPRXIT) — Terminates AMDPRDMP.
  - AMDPRRDC (AMDPRRDA) — Free buffers.
  - AMDPRUIM (AMDUSRDL) — Calls BLSQEXTI to delete exit services.

Formatting Statement Processing (Diagrams 6-10)

- AMDPRCTL (AMDPRMSC) — Scans each control statement and determines processing path.
  - AMDPRCOM (AMDPRSYN) — Writes syntax error messages.
  -
    - AMDPRUIM (AMDUSRXT) — Invokes BLSQECT to link to the user module corresponding to the user verb, for example, SRMDATA, MTRACE or ASMDATA.
      - AMDPRRDC (AMDPRRDC) — INIT function: ensures initialization for dump has been performed.
      - BLSQEXTI — Load exit service and initialize ABDPL and SRA.
      - BLSQROUT — To invoke BLSQECT to link to exit module.
      - AMDPRRDC (AMDPRRDD) — DATA function: provides data for exit module.
      - AMDPRFMT — Provides format service for exit module.
    - AMDPRSEG — Loads processing path for LPAMAP, EDIT, SUMMARY, CPUDATA, or PRINT statements; calls for each module to store its entry addresses in COMMON.
      - AMDPRRDC (AMDPRRDC) — INIT function: ensures initialization for dump is performed.
  - Executor module for statements processed by AMDPRSEG.

CPUDATA (Diagram 10)

AMDPRCTL (AMDPRMSC) — Scans control statement.

- AMDPRGCD — Executor: Prints general data for each CPU including: all registers, the current PSW, logical and physical CCA, the current PCLINK STKE queue, the PSA, and a one line indication of what the four cross memory ASIDs are. Loads format modules IEEVCSD, IEALCCP, IEEVPCCA, IEAPSAP, and IEASVTP.
  - AMDPRRDC (AMDPRRDD) — Locates data in the dump.
    - AMDPREAD — Reads data into buffers.
  - AMDPRCOM (AMDPRGFR, AMDPRWTR) — Prints title. Formats and writes output.

Verb Exits Defined in the ECT (Diagram 7)

AMDPRUIM (AMDUSRXT) — Initializes exit interface. Provides storage access routine (AMDMEMAR), write routine (AMDWRITR, and index routine (AMDINDEX) for user exits. Formats and prints requested information.

- User exit
- AMDPRRDC (AMDPRRDD) — Locates data for user exit.
  - AMDPREAD — Reads records into buffers.
- AMDPRFMT — Formats output.
- AMDPRCOM (AMDPRWTR) — Writes output.
- AMDPRCOM (AMDPRNDX) — Writes index entries.

PRINT STORAGE/REAL (Diagram 9)

AMDPRCTL (AMDPRMSC) — Scans control statement.

- AMDPRPMS — Executor: Determines storage ranges to be printed from PRINT parameters.
  - AMDPRCOM (AMDPRWTR) — Writes print title.
  - AMDPRDPS (AMDPRPCB, PRNTSTG) — Builds and enqueues PCBs to describe storage ranges for printing. Prints storage using PCBs.
    - AMDPRRDC (AMDPRRDD) — Locates dump records as required.
      - AMDPREAD — Reads records into buffers.
    - AMDPRCOM (AMDPRWTR) — Writes output.

PRINT NUCLEUS/SQA/CSA (Diagram 9)

AMDPRCTL (AMDPRMSC) — Scans control statement.

- AMDPRNUC — Executor: Controls printing of nucleus, system queue area, and common service area.
  - AMDPRCOM (AMDPRWTR) — Writes nucleus title.
  - AMDPRUIM (AMDRUSRXT) — (NUCLEUS only) Interfaces with user's print nucleus exit.
  - AMDPRDPS (AMDPRPCB, PRNTSTG) — Builds and queues PCBs for storage to be printed. Prints storage.
    - AMDPRRDC (AMDPRRDD) — DATA function: Locates records.
      - AMDPREAD — Reads records into buffers.
    - AMDPRCOM (AMDPRWTR) — Writes output.

PRINT CURRENT (Diagram 10)

AMDPRCTL (AMDPRMSC) — Scans control statement.

- AMDPRPCR — Executor: controls formatting and printing of control blocks and of region for current task.
  - AMDPRCOM (AMDPRWTR) — Writes CURRENT title.
  - AMDPRUIM (AMDUSRXT) — Interfaces with CURRENT/JOBNAME exit.
  - AMDPRFSC — Controls formatting and printing of control blocks related to address space and task.
  - AMDPRFUB — Determines region boundaries for building PCBs.
  - AMDPRUIM (AMDUSRXT) — Interfaces with the SUMMARY FORMAT verb exit to format control blocks related to address space and task.
  - AMDPRDPS (PRNTSTG) — Prints user region specified by PCBs.
    - AMDPRRDC (AMDPRRDD) — Locates dump records.
      - AMDPREAD — Reads records into storage.
    - AMDPRCOM (AMDPRWTR) — Writes output.
  - AMDPRFSR (TCBSMRY) — Writes TCB summary.

PRINT JOBNAME (Diagram 10)

- AMDPRCTL (AMDPRMSC) — Scans control statement.
  - AMDPRJB — Executor: scans jobname in control statement and builds jobname lists; controls formatting and printing of control blocks and storage for jobnames.
    - AMDPRCOM (AMDPRSYN, AMDPRWTR) — Writes syntax error messages. Writes output.
    - AMDPRUIM (AMDUSRXT) — Interfaces with CURRENT/JOBNAME exit. module.
    - AMDPRRDC (AMDPRRDD) — Locates data in dump.
      - AMDPREAD — Reads required records into buffers.
    - AMDPRFSC — Controls formatting of system control blocks for each jobname.
    - AMDPRFUB — Determines region boundaries for ASCBs.
    - AMDPRIUM (AMDUSRXT) — Interfaces with the SUMMARY FORMAT verb exit to format control blocks related to address space and task.
    - AMDPRDPS (PRNTSTG) — Formats and controls printing of storage areas.
      - AMDPRRDC (AMDPRRDD) — Locates data in dump.
        - AMDPREAD — Reads records into buffer.
      - AMDPRCOM (AMDPRWTR) — Writes output.
    - AMDPRFSR (TCBSMRY) — Prints TCB summary.

EDIT (Diagram 12)

AMDPRCTL (AMDPRMSC) — Scans EDIT statement.

- AMDPRSCN — Scans EDIT parameters.
  - AMDPROOT — Initializes AMDPRTAB communications table.
  - AMDPRSN2 — Processes EDIT parameters.
  - AMDPRCOM (AMDPRSYN) — Writes syntax error messages.
  - AMDPRSN3 — Processes statement end. Sorts range tables, checks options.
- AMDPRFRM — Controls editing/printing of GTF data.
  - AMDPRCOM (AMDPRWTR, AMDPRSYN, AMDPRMSG) — Writes EDIT output. Writes syntax error messages. Writes output messages.
  - AMDPRGET — Obtains GTF record for processing.
    - AMDPRCOM (AMDPRWTR, AMDPRSYN) — Writes output or syntax error messages.
    - AMDPRRDC — Locates required records.
      - AMDPREAD — Reads records into buffer.
- AMDPRREC (AMDPRCON) — Formats GTF control records.
  - AMDPRCOM (AMDPRWTR, AMDPRGFR) — Formats an output line. Writes the formatted line.
- AMDPRFLT — Checks whether options specify processing the current record.
  - AMDPREXT — User exit interface.
    - AMDPRCOM (AMDPRSYN, AMDPRWTR) — Syntac error messages. Writes output.
    - AMDPRSEG — Loads user exit module.
    - AMDPRREC (AMDPRHEX) — Formats AID, FID, AND EID fields and writes the fields in hexadecimal.
  - User exit routines.
- AMDPRAPP — Interfaces with format appendage.
  - AMDPRCOM (AMDPRSYN, AMDPRWTR, AMDPRGFR) — Hexadecimal conversion routine, formats and prints an output line.
  - AMDPRSEG — Loads the format appendage.
  - AMDPRREC (AMDPRHEX) — Formats the AID, FID, and EID fields of the record. Writes the fields in hexadecimal.
    - AMDPRCOM (AMDPRGFR, AMDPRWTR) — Formats an output line. Writes the formatted line.
  - Format appendage

Section 4: Data Areas

This section describes the following data areas that are built and used by AMDPRDMP modules:

- Print dump index description table (AMDMNDXT).
- ASCB map (ASCBMAP).
- ASID index (ASIDNDX).
- Buffer map entry.
- Common communication area (COMMON).
- CPU status area.
- Current TCB list (CURRLIST).
- Dump map entry.
- EDIT communication table (AMDPRTAB).
- Exit parameter list (ABDPL).
- Exit control table (ECT) entry.
- Exit control table entry mapping (ECTE).
- Mapping for dump header and processor status input records (AMDDATA).
- Path descriptor element.
- Print control block (PCB).
- TCBLIST entry.

Address Space Control Block Map (ASCBMAP)

Size: Variable; depends on number of ASIDs. For each ASID, there is a map entry and table entry.

Map — 16 static bytes — 2 bytes per active ASID (rounded up to double word boundary.)

Table — 8 bytes per active ASID.

Created by: AMDPRRDC

Use: Map contains the ASID of each address space on ASCB dispatching queue at time of dump. Table contains the corresponding ASCB address and segment table address for each ASID in map.

Map Entry

Offset		Size	Field Name	Description
0	(0)	4	ASCBTAB	Pointer to the table of addresses of ASCBs and segment tables.
4	(4)	2	ASCBNUM	Number of address spaces in the table.
6	(6)	2	ASCBRESV	Reserved.
8	(8)	4	ASMAPLEN	Length of ASCB map.
12	(C)	4	ASTABLEN	Length of ASCB/SEGTAB table.
16	(10)	2	ASID	First address space ID in table.
xx	(xx)	2	- - - -	Last ASID in table.

Table Entry

Offset		Size	Field Name	Description
0	(0)	4	ASCBADDR	Address of ASCB associated with first ASID in map. X'80' in the high order byte indicates AMDPRRDC was unable to read corresponding ASID for this ASCB. ASID field is set to FFFF.
4	(4)	4	SEGTABAD	Real address of segment table associated with first ASID in map.

Exit Parameter List (ABDPL)

Size: 96 bytes

Created by: AMDPRUIM

Use: The exit parameter list serves as a communications area between AMDPRDMP, user exit routines, and the AMDPRDMP service routines. The BLSADPL macro maps the exit parameter list. The exit parameter list is created by AMDPRDMP, IPCS and SNAP with the intent of interfacing to exit routines. For more detailed information on the mapping, see MVS/XA IPCS Logic & Diagnosis. (The IHAABDPL is still available for compatibility. It includes the BLSABDPL macro.)

Address Space Identifier Index (ASIDNDX)

Size: 112 bytes per table.

Created by: AMDPREAD or AMDPRLOD.

Use: The index is searched for the requested ASID from which the corresponding map address is found (used to find the block containing the requested address).

Offset		Size	Field Name	Description
0	(0)	4	ASDXLNK	Pointer to next table or zero.
4	(4)	36	ASDXASID	18 2-byte fields, each of which contains an ASID.
40	(28)	72	ASDXMAP	18 4-byte fields, each of which contains the address of the first dump-data map for the corresponding ASID.

Buffer Map Entry

Size: 20 bytes.

Created by: AMDPRRDC.

Updated by: AMDPRRDC, AMDPREAD.

Use: One buffer map entry is built for each AMDPRDMP buffer.

Offset		Size		Field Name	Description
0	(0)	4		BUFFLINK	Address of next buffer map entry in chain or zero if last entry. Buffer map entries re chained in order from most-recently referenced buffer to least-recently referenced buffer.
4	(4)	4		BUFFPTR	Address of an input buffer.
8	(8)	4		BUFFREAL	Address of real storage location of the 4K block of storage contained in the dump record that was read into the associated buffer.
12	(C)	4		BUFFVIRT	Address of virtual storage location of the 4K block of storage contained in the dump record that was read into the associated buffer.
16	(10)	2		BUFFASID	ASID of data in buffer.
18	(12)	2		BUFFCPU	Logical CPU ID if data is a CPU status record.
20	(14)	1		BUFFLAG	Flags:
		1...	....	INVALFLG	Invalid data in the buffer.
		.1..	....	BUFFCOM	ASID of data is FFFF.
		..xx	xxxx	----	Reserved.

Common Communication Area (COMMON)

Size: 2884 bytes.

Created by: AMDPRCOM.

Used by: All AMDPRDMP modules.

Use: Communication among AMDPRDMP modules.

Offset		Size	Field Name	Description
0	(0)	4	ERRADDR	Address of current error routine to receive control if a read error occurs.
4	(4)	4	VERBGN	Address of verb in current control statement.
8	(8)	4	VERBEND	Address of first character following verb in current control statement.
12	(C)	4	KYWDBGN	For PRINT control statement only, address of keyword.
16	(10)	4	KYWDEND	For PRINT control statement only, address of first character following keyword.
20	(14)	4	DELIMCD	Code Delimiter: X'04' Comma. X'08' Equal sign. X'0C' Blank. X'10' Left parenthesis. X'14' Right parenthesis.
24	(18)	8	WORK1	Doubleword work area.
32	(20)	4	SIX	Fullword constant: 6.
36	(24	4	LINECNT	Current line number, used to determine when a new page is needed.

Offset		Size		Field Name	Description
40	(28)	4		PAGENUMB	Current page number in binary.
44	(2C)	4		CURBUF	Address of current output buffer supplied by AMDPRWTR.
48	(30)	4		TCBLIST	Origin of TCB list.
52	(34)	4		CVTADDR	Address of CVT in dump system, obtained from the dump system or from the CVT control statement.
56	(38)	4		PCBPTR	Origin of PCB queue.
60	(3C)	8		INDD	DDNAME of input data set DD statement, set by the NEWDUMP control statement or default of TAPE.
68	(44)	4		RDENTRY	Count of DATA entries to module AMDPRRDC, printed as summary information at AMDPRDMP termination.
72	(48)	4		READNO	Number of blocks read from SYSUT1 work data set or dump tape, printed as summary information at AMDPRDMP termination.
76	(4C)	4		RDERCNT	Number of I/O errors while attempting to read from SYSUT1 work data set or dump tape, printed as summary information at AMDPRDMP termination.
80	(50)	4		READTM	Number of times data requested was not in an input buffer, printed as summary information at AMDPRDMP termination.
84	(54)	2		FILESEQ	Value of FILESEQ parameter in NEWDUMP control statement, used to open the input tape to the correct file.
86	(56)	2		ONEA	Constant: 1.
88	(58)	2		TWO	Constant: 2.
90	(5A)	2		THREE	Constant: 3.
92	(5C)	2		FOUR	Constant: 4.
94	(5E)	2		FIVE	Constant: 5.
96	(60)	2		SEVEN	Constant: 7.
98	(62)	2		EIGHT	Constant: 8.
100	(64)	2		TEN	Constant: 10.
102	(66)	2		ELEVEN	Constant: 11.
104	(68)	2		TWELVE	Constant: 12.
106	(6A)	2		SIXTEEN	Constant: 16.
108	(6C)	2		LINENUMB	Value of LINECNT parameter in PARM field of EXEC statement or default of 58.
110	(6E)	2		LINENUM	Value of LINENUMB minus 2. (The number of lines per page excluding the title).
112	(70)	1		RETCODE	Return code for PRDMP step.
113	(71)	1		SWA	Switches:
		1...	....	SYSUT1SW	SYSUT1 data set specified.
		.1..	....	SYSUT2SW	SYSUT2 data set specified.
		..1.	....	OPSSW	Operands buffering in process.
		...1	....	NOTPAGES	Title pages are not to be repeated after 'CVT=' control statement.
		....	1...	GOSW	GO control statement being processed.
		....	.1..	DDSW	SYSPRINT DD statement specified.
		....	..1.	SETCVTSW	CVTADDR field in COM has been filled by CVT control statement.
		....	...1	RDRSW	Control statements entered from the SYSIN data set: otherwise, from the console.
114	(72)	1		SWB	Switches:
		1...	....	IOERR	I/O error while reading from SYSUT1 or dump tape.
		.1..	....	FMTERR	Requested information not in input data set.
		..1.	....	PRTSUM	SUMMARY called from PRINT.
		..x.	....	----	Reserved.
		....	1...	ENDSW	END control statement being processed.
		....	.1..	QSYSUT2	SYSUT2 data set specified.
		....	..xx	----	Reserved.

Offset		Size		Field Name	Description
115	(73)	1			SWC Switches:
		1...		....	MSTRSW AMDPRMST has attempted dump initialization.
		.1..		....	SETFLSH Set FLSHMODE in SWD if error occurs.
		..1.		....	POSITSW Positioning of the input data set must be performed. When the main error handler in AMDPRCTL is entered and this switch is on, an error occurred in input data set positioning.
		...1		....	TREADIN INDD data set on direct access.
		....		1...	EDITSW EDIT control statement being processed.
		....		.1..	SEGRD EDIT Module indication to loader module to issue an initialization read.
		....		..xx	---- Reserved.
116	(74)	1			SWD Switches:
		1....		....	FLSHMODE Control statements are being scanned for syntax errors only.
		.xx.		....	---- Reserved.
		...1		....	NOSTDMG Do not issue message AMD161I before servicing caller's request.
		....		1...	NOLOADSW Another control statement appeared before END control statement.
		....		.1..	CONTSW Obtain continuation statement for EDIt control statement.
		....		..1.	GPRSFND Issue CPUSTATUS BRREAD to print general register contents in output listing.
		....		...x	---- Reserved.
117	(75)	1			SWE Switches:
		1...	....	STOPSW	Stop option specified.
		.x..	....	----	Reserved.
		..1.	....	RESPC	Respecify EDIT options
		...1	....	TITLESW	Title option specified.
		....	1...	BUILDMAP	AMDPRLOD or AMDPREAD must scan dump to build map(s).
		....	.xxx	----	Reserved.
118	(76)	1		SWF	Switches:
		1...	....	QPRDINIT	AMDPREAD must be specified.
		.x..	....	----	Reserved.
		..1.	....	QSADMP	Input is from AMDSADMP.
		...1	....	DMPIC	Complete dump contained in buffers.
		....	1...	PAGEOK	Switch for AMDPRFCB.
		....	.x..	----	Reserved.
		....	..1.	QUT1LOD	SYSUT1 is preloaded.
		....	...x	----	Reserved.
119	(77)	1		BUFSW	Switches:
		xxxx	xx..	----	Reserved.
		....	..1.	PREFM	Preformatted dump tape being processed.
		....	...1	SUMFSRSW	Communicate AMDPRSUM and AMDPRFSR. Switch.
120	(78)	1		PRSW	Switches:
		1...	....		Initialize exit interface for mainline print dump request.
		.1..	....	QSEGTBSW	Segment table origin supplied by user in SEGTAB control statement.
		..x.	....	----	Reserved.
		...1	....	TTLSW	SVC dump header record has been printed in PRINTER output data set.
		....	1...	PRNTRL	PRINT REAL is current control statement.
		....	.1..	PRNTREAL	Real storage data requested.
		....	..xx	----	Reserved.
121	(79)	1		INDEXSW	Dump index switch.
122	(7A)	3		----	Reserved.
125	(7D)	81		WTORMSG	Control statement input area.

Offset		Size	Field Name	Description
206	(CE)	8	BLANKS	Character string constant blanks.
214	(D6)	5	TITLE	Character string constant: "TITLE".
219	(DB)	4	STOP	Character string constant: "STOP".
223	(DF)	26	MSG1	ID and skeleton of message AMD161I.
249	(F9)	26	MSG2	ID and skeleton of message AMD158I.

The following nine fields (to offset X'18F') define the AMDPRDMP page title.

Offset		Size	Field Name	Description
275	(113)	64	TITLEMSG	Title area for user-supplied title.
339	(153)	1	----	One blank.
340	(154)	15	TITLEMOD	Label and field for module name from dump data set header record.
355	(163)	1	----	One blank.
356	(164)	13	TITLEDTE	Label and field for data on which dump was taken.
369	(171)	2	----	Two blanks.
371	(173)	13	TITLETME	Label and field for time at which dump was taken.
384	(180)	2	----	Two blanks.
386	(182)	13	TITLEPGE	Label and field for page number.
399	(18F)	256	CAPTABL	Translate table for translating EBCDIC lower case to upper case.
655	(28F)	256	TABLE	Translate table to form EBCDIC printout at right margin of general format.
911	(38F)	63	HEXTABL	Translate table for first stage of conversion of binary to hexadecimal characters. *Note:* TABLE, HEXTABL, and EBCTABL overlap to save space.
974	(3CE)	10	EBCTABL	Translate table for second stage of conversion of binary to hexadecimal characters. *Note:* TABLE, HEXTABL, and EBCTABL overlap to save space.
984	(3D8)	256	BLNK	Main scan table. This translate table associates the following codes with stopping characters: **Code Delimiter** X'04' Comma X'08' Equal sign X'0C' Blank X'10' Left parenthesis X'14' Right parenthesis
1240	(4D8)	256	NONBLNK	Table for nonblank character scan. This table recognizes only alphanumerics as nonblank.
1496	(5D8)	256	NONBLANK	Table for nonblank character scan. This table recognizes everything except X'40' as nonblank.
1752	(6D8)	4	AWRITE	Address of PRINTER data set write routine, AMDPRWTR.
1756	(6DC)	4	APRTMSG	Address of SYSPRINT data set write routine, AMDPRMSG.
1760	(6E0)	4	ASYNTAX	Address of message writer routine, AMDPRSYN.
1764	(6E4)	4	AFMTLINE	Address of general format routine, AMDPRGFR.
1768	(6E8)	4	AADRCNVT	Address of 3-byte binary to hexadecimal conversion routine in CSECT AMDPRSYN.
1772	(oEC)	4	AWRDCNVT	Address of 4-byte binary to hexadecimal conversion routine in CSECT AMDPRSYN.

Offset		Size		Field Name	Description
1776	(6F0)	4		ARGNBND	Address of AMDPRFUB, valid only when the following control statements are being processed: PRINT ALL. PRINT CURRENT. PRINT JOBNAME.
1780	(6F4)	4		STOPEXIT	Address of current stop-exit routine. This routine is invoked if the STOP option of AMDPRDMP is in effect and the operator replies 'STOP' to message AMD156I.
1784	(6F8)	4		SYNMSGA	Address of message address list which contains messages that can be issued by calling AMDPRSYN at each of its entry points.
1788	(6FC)	4		AEREXIT	Address of AMDPRDMP termination routine, AMDPRXIT.
1792	(700)	4		ALOADER	Address of AMDPRSEG.
1796	(704)	4		QATMERTN	Address of TODCNVRT in AMDPRSEG.
1800	(708)	4		ATCBSAVE	Address of TCB list enqueue routine in CSECT AMDPRTSV of AMDPRCOM.
1804	(70C)	4		ATCVREMV	Address of TCB list dequeue routine in CSECT AMDPRTSV in AMDPRCOM.
1808	(710)	4		ATCBRTRV	Address of TCB list element retrieve routine in CSECT AMDPRTSV of AMDPRCOM.
1812	(714)	4		APCBENQ	Address of PCB queue enqueue routine, valid only when the PRINT with all keywords control statements are being processed.
1816	(718)	4		ASTPROUT	Address of STOP option handler. CSECT AMDPRSTP in AMDPRCOM.
1820	(71C)	4		AFORMAT	Address of AMDPRFSR, valid only when the following control statements are being processed. FORMAT. PRINT CURRENT. PRINT JOBNAME.
1824	(720)	4		APRTSTG	Address of AMdPRDPS, valid only when the PRINT or CPUDATA control statements are being processed.
1828	(724)	4		BUFSUM	Total number of dump data set buffers used during this execution of AMDPRDMP.
1832	(728)	4		BUFREINT	Number of times dump data set buffers have been initialized during this execution of AMDPRDMP.
1836	(72C)	4		AEND	Address of END control statement processing routine AMDPREND.
1840	(730)	4		ONGOOPTR	Address of operands specified in the ONGO control statement, used by the GO verb to determine the functions to be performed.
1844	(734)	4		----	Reserved.
1848	(738)	24		DCBADDRS	Origin of AMDPRDMP DCB address array.
1848	(738)	4		AOUTDCB	Address of PRINTER data set DCB.
1852	(73C)	4		ANDXDC3	Address of INDEX data set DCB.
1856	(740)	4		APTRDCB	Address of SYSPRINT data set DCB.
1860	(744)	4		ARDRDCB	Address of SYSIN data set of DCB.
1864	(748)	4		AINDCB	Address of input data set DCB.
1868	(74C)	4		----	Alignment.
1868	(74C)	1			
		1...	....	ENDLIST	Constant X'80' to indicate end of DCB address array.
1869	(74D)	3		ASYSUDCB	Address of SYSUT1 or SYSUT2 data set DCB.

The following four fields (to offset X'75D') are used only for the EDIT function of AMDPRDMP. These fields are valid only when the EDIT control statement is being processed.

Offset		Size	Field Name	Description
1872	(750)	4	TRCCOUNT	Number of GTF trace records processed, i.e., the number of entries to module AMDPRGET.
1876	(754)	4	AEDITCB	Address of the EDIT communications area AMDPRTAB.
1880	(758)	4	AROOT	Address of AMDPRTAB initialization routine AMDPROOT.
1884	(75C)	1	EDITER	Value of EDIT ER = parameter from AMDPRDMP EXEC statement.
1885	(75D)	3	----	Alignment.

Additions for release one support.

Offset		Size	Field Name	Description
1888	(760)	4	REALMAP	Address of real storage dump map.
1892	(764)	4	SEGTABOR	Segment table origin.
1896	(768)	4	REALMAX	Address of the top of real storage.
1900	(76C)	4	QAPFT	Address of page frame table.

Additions for release one support.

Offset		Size	Field Name	Description
1904	(770)	2	QASID	ASID of IPLed CPU for SADMP; home ASID for SVC dump.
1906	(772)	2	IPLCPU	Address of IPLed CPU. Contains 256 if CPU status was unavailable; that is, an I/O error occurred while reading the CPUSTATUS record.
1908	(774)	4	CURASCB	Address of current ASCB in dumped system.
1912	(778)	4	PREFXRGR	Real address in PSA prefix register (SADMP input only).
1916	(77C)	4	PREFXRGV	Virtual address in PSA prefix register (SADMP input only.)
1920	(780)	4	ASVTADDR	Address of ASVT in dumped system.
1924	(784)	168	HDRREGS	Registers and current PSW from SVC Dump or DSS header record.
2092	(82C)	100	HDRTITLE	Title from dump header record.
2192	(890)	4	ASIDNDX	Address of ASID index.
2196	(894)	4	CPUMAP	Address of CPU STATUS record maps.
2200	(898)	4	ASCBMAP	Address of ASCB map created by AMDPRRDC.
2204	(89C)	4	BUFERMAP	Address of first physical buffer map entry in AMDPRRDC.
2208	(8A0)	4	BRRDDATA	Address of AMDPRRDC data read routine.
2212	(8A4)	4	BRRDINIT	Address of AMDPRRDC INIT routine.
2216	(8A8)	4	BRRDADJ	Address of BRREAD ADJUST routine.
2220	(8AC)	4	AASCBFMT	Address of AMDPRFAR.
2224	(8B0)	4	ASRBFMT	Address of AMDPRSRB.
2228	(8B4)	4	AUSRINIT	Address of AMDPRUIM initialization routine.
2232	(8B8)	4	AUSREXIT	Address of user-exit interface routine.
2236	(8BC)	4	AUSRDEL	Entry point of AMDPRUIM clean-up routine.
2240	(8C0)	4	AUSRTCBA	Address of TCB currently being processed by AMDPRFSR.
2244	(8C4)	2	AUSRASID	ASID of address space being processed by AMDPRFAR or AMDPRFSR.
2246	(8C6)	1	EXITFLAG	Flags indicating action to be performed by AMDPRUIM.

The following offsets are used in the IBM 3800 printing subsystem:

Offset		Size		Field Name	Description
2247	(8C7)	1		I3800SW	Switches:
		xxxx	....	----	Reserved.
		....	1...	I3800KEY	Indicates storage key message will be printed in page title.
		....	.1..	I3800ULN	User specified line count.
		....	..1.	I380080	Print 80 lines per page, subject to user specified line count.
		....	...1	I3800204	Print double density line for storage dump.
2248	(8C8)	2		RBMAX	Maximum number of RBs allowed (50).
2250	(8CA)	2		LLEMAX	Maximum number of LLEs allowed (255).
2252	(8CC)	2		JPQMAX	Maximum number of PQEs (256).
2254	(8CE)	2		DEBMAX	Maximum number of DEBs (200).
2255	(8D0)	2		DDMAX	Maximum number of DDs (1635).
2258	(8D2)	2		SRBMAX	Maximum number of SRBs (50).
2260	(8D4)	2		TCBMAX	Maximum number of TCBs (256).
2262	(8D6)	2		ASCBMAX	Maximum number of ASCBs (200).
2264	(8D8)	4		XLMAX	Maximum number of extent lists (25).
2268	(8DC)	4		LPAMAX	Maximum number of CDE elements (500).
2272	(8E0)	16		TITLEKEY	Label and field for key or storage data currently being dumped.
2272	(8E0)	12		----	Reserved for alignment.
2284	(8EC)	2		TITLESTK	Storage key value.
2288	(8F0)	12		Z9ERRID	Stored from the SVC dump.
2300	(8FC)	4		APRSTK	Address of the STKE formatter.
2304	(900)	24		CMINFO	Information about cross memory environment.
2304	(900)	8		CMASID	Current ASIDs.
2304	(900)	2		CMHASID	ASID of the HOME address space.
2306	(902)	2		CMPASID	ASID of the PRIMARY address space.
2308	(904)	2		CMSASID	ASID of the SECONDARY address space.
2310	(906)	2		CMCASID	ASID of the CML (cross memory lock) address space.
2312	(908)	16		CMASCB	Current ASCBs.
2312	(908)	4		CMHASCB	ASCB pointer to the HOME address space.
2316	(90C)	4		CMPASCB	ASCB pointer to the PRIMARY address space.
2320	(910)	4		CMSASCB	ASCB pointer to the SECONDARY address space.
2324	(914)	4		CMCASCB	ASCB pointer to the CML address space.
2328	(918)	8		CMPSW	PSW passed from SVCDUMP and SYSMDUMP.
2336	(920)	1		CMCFLGS	Flags needed by AMDPRCMC.
		1...	....	CMDFTCUR	Above ASID and ASCB fields should not be used by PRINT CURRENT and PRINT STORAGE verbs.
		.xxx	xxxx	----	Reserved.
2337	(921)	3		----	Alignment.
2340	(924)	4		CMNUMPCB	The number of PCBs to be processed and passed from AMDPRPMS to AMDPRDPS.
2344	(928)	200		COMSRES	Reserved.
2520	(9D8)	16		PDCPPL	TSO CPPL.
2520	(9D8)	4		PDCBUF	Address of TSO command buffer.
2524	(9DC)	4		PDCUPT	Address of TSO UPT.
2528	(9E0)	4		PDPSCB	Address of TSO PSCB.
2532	(9E4)	4		PDPECT	Address of TSO ECT.
2536	(9E8)	4		ABDPLADR	Address of TSO ABDPL.
2540	(9EC)	4		SRAADR	Address of Service Router Area, (SRA).
2544	(9F0)	4		COMTRCE	Anchor for the Console-Initiated Loop Recording routine.
2548	(9F4)	1		PDSW	Switches:
		1...	....	PSLITERM	AMDPRDMP processing must terminate.
		.1..	....	PSQASPIL	The SQA has spilled into the CSA.
		..1.	....	POVERLAP	The storage ranges overlap when enqueuing PCBs.
		...1	....	PGDAERSW	The GDA is unavailable.
		....	xxxx	----	Reserved.

Offset		Size	Field Name	Description
2549	(9F5)	1	----	Reserved.
2550	(9F6)	2	FBQEMAX	Maximum number of FBQEs (100).
2552	(9F8)	124	PSLIDTA	Data initialized by AMDPRSLI.
2552	(9F8)	8	PCSADINF	CSA starting addresses.
2552	(9F8)	4	PCSADDR	Non-extended CSA starting address.
2556	(9FC)	4	PCSAEADR	Extended CSA starting address.
2560	(A00)	8	PCSZINFO	CSA size information.
2560	(A00)	4	PCSASZ	Non-extended CSA size.
2564	(A04)	4	PCSAESZ	Extended CSA size.
2568	(A08)	8	PSQADINF	SQA starting address.
2568	(A08)	4	PSQADDR	Non-extended SQA starting address.
2572	(A0C)	4	PSQAEADR	Extended SQA starting address.
2576	(A10)	8	PSQZINFO	SQA size information.
2576	(A10)	4	PSQASZ	Non-extended SQA size.
2580	(A14)	4	PSQAESZ	Extended SQA size.
2584	(A18)	8	PRIADINF	Private starting address.
2584	(A18)	4	PRIADDR	Non-extended private starting address.
2588	(A1C)	4	PRIEADR	Extended private starting address.
2592	(A20)	8	PRITAINF	Private ending addresses.
2592	(A20)	4	PRITADDR	Non-extended private ending address.
2596	(A24)	4	PRITEADR	Extended private ending address.
2600	(A28)	8	PRISZINF	Private size information.
2600	(A28)	4	PRISZ	Non-extended private size.
2604	(A2C)	4	PRIESZ	Extended private size.

The following represents the address of the GDA fields for which BRREADS were attempted. The address of the GDA fields is used in an output comment when a read error occurs.

Offset		Size	Field Name	Description
2608	(A30)	8	PCSERRAD	Address of GDA fields for CSA addresses.
2608	(A30)	4	PRDCSAD	Address of GDA field for non-extended CSA addresses.
2612	(A34)	4	PRDCSADE	Address of GDA field for extended CSA starting address.
2616	(A38)	8	PCSERRSZ	Address of GDA fields for CSA size.
2616	(A38)	4	PRDCSASZ	Address of GDA field for non-extended CSA size.
2620	(A3C)	4	PRDCSZE	Address of GDA field for extended CSA size.
2624	(A40)	8	PSQERRAD	Addresses of GDA fields for SQA addresses.
2624	(A40)	4	PRDSQAD	Address of GDA field for non-extended SQA starting address.
2628	(A44)	4	PRDSQADE	Address of GDA field for extended SQA starting address.
2632	(A48)	8	PSQERRSZ	Address of GDA fields for SQA size.
2632	(A48)	4	PRDSQASZ	Address of GDA field for non-extended SQA size.
2636	(A4C)	4	PRDSQSZE	Address of GDA field for extended SQA size.
2640	(A50)	4	PNUCLBND	Lowest DAT-on nucleus address.
2644	(A54)	4	PNUCTOP	Highest DAT-on nucleus address.
2648	(A58)	4	PNUCRLBD	Lowest DAT-off nucleus address.
2652	(A5C)	4	PNUCRTOP	Highest DAT-off nucleus address.
2656	(A60)	4	PDFLTNCL	Default for lowest DAT-on nucleus address.
2660	(A64)	4	PDFLTNCT	Default for highest DAT-on nucleus address.
2664	(A68)	12	PAQATNDX	Pointers to the SQA AQAT index tables for subpools 226, 239, and 245.

Offset		Size	Field Name	Description
2264	(A68)	4	PAQNDX26	Pointer to the subpool 226 AQAT index table.
2268	(A6C)	4	PAQNDX39	Pointer to the subpool 239 AQAT index table.
2272	(A70)	4	PAQNDX45	Pointer to the subpool 245 AQAT index table.
2676	(A74)	15	TOPHDR	Print storage heading.
2691	(A83)	1	----	Alignment.
2692	(A84)	20	PSLIDTA1	Data initialized by AMDPRSLI.
2692	(A84)	4	PMAXCOME	Maximum ending address of extended common area when comparing the end of the extended FLPA, MLPA, PLPA, CSA, and SQA.
2969	(A88)	4	PSMAD	Start of the non-FREEMAINable storage management area.
2700	(A8C)	4	PSMSZ	Size of the non-FREEMAINable storage management area.
2704	(A90)	4	PPGTAD	Start of the non-FREEMAINable page table area.
2708	(A94)	4	PPGTSZ	Size of the non-FREEMAINable page table area.
2712	(A98)	4	COMHDR	Pointer to a copy of the dump data set header record.
2716	(A9C)	44	COMDSNM	Work file data set name from AMDPRLOD.
2760	(AC8)	4	COMPRFOR	Address of entry in AMDPRFOR to be used by AMDPRFXT during format verb processing.

The following field is an addition for 4K read support.

Offset		Size	Field Name	Description
2764	(ACC)	4	COM4KBUF	Address of 4K buffer used in 4K BRREAD.

The following two fields are additions for input record continuation support.

Offset		Size	Field Name	Description
2768	(AD0)	4	COMOPLN	Length of operands list.
2772	(AD4)	4	COMOPPTR	Pointer to operands list.

The following six fields (to offset X'AEB') are used for AMDPRDMP index support.

Offset		Size	Field Name	Description
2776	(AD8)	4	COMINDXA	Address of index entry.
2780	(ADC)	1	COMINDEX	Entry code.
2781	(ADD)	1	COMINDXL	Index level.
2782	(ADE)	2	COMXASID	Insertion data for index.
2784	(AE0)	4	COMNDXEP	Entry point for AMDPRNDX, the dump index service routine.
2788	(AE4)	4	INDXLCNT	Index line count work area.
2792	(AE8)	4	VIRTMAX	Highest virtual address. This field is set by AMDPRSLI from CVTMZ00, if it can be read from the dump.
2796	(AEC)	16	PDIOPL	TSO IOPL.
2796	(AEC)	4	PDIUPT	Address of TSO UPT.
2800	(AF0)	4	PDIECT	Address of TSO ECT.
2804	(AF4)	4	PDIECB	Address of ECB.
2808	(AF8)	4	PDIIOPBP	Address of I/O parameter block.

CPU Status Area

Size: 192 bytes.

Created by: AMDPRRDC.

Use: A BRREAD request for CPU status will result in this area being created from the input dump header record or CPU status records. The IHAABDLP macro contains mapping for this information.

Offset		Size		Field Name	Description
0	(0)	1		AMDCFLAG	
		1...	....		CPU is a uniprocessor: CPU address is invalid.
		.1..	....		SADMP unable to perform store status; only CPU address is valid.
		..1.	....		Operator did not perform store status; only general registers and, if MP, CPU address are valid.
		...1	....		Non-SADMP input, only registers/PSW are valid.
		....	xxxx		Reserved.
1	(1)	1			Reserved.
2	(2)	2		AMDCPADR	CPU address.
4	(4)	32		AMDCFREG	Floating point registers 0-6.
46	(24)	64		AMDCGREG	General purpose registers 0-15.
100	(64)	64		AMDCCREG	Control registers 0-15.
164	(A4)	8		AMDCCPSW	Current PSW.
172	(AC)	4		AMDCPREG	Prefix register value.
176	(B0)	8		AMDCTIME	CPU timer value. **Note:** This is not location 80 time or TOD clock.
184	(88)	8		AMDCLOCK	Clock comparator value.

Current TCB List (CURRLIST)

Size: 192 bytes.

Created by: AMDPRSUM.

Use: SUMMARY control statement uses this data area for storing information for the processor that is not in SRB mode.

Offset		Size	Field Name	Description
0	(0)		CURENTRY	List entries —one for each current TCB found in the dump.
0	(0)	2	CURCPUID	CPU identifier for this entry.
2	(2)	2	----	Reserved.
4	(4)	4	CURTCBA	TCB address for this entry.
8	(8)	4	CURASCA	ASCB address for this entry.

Dump Header Record

The Dump Header Record is described in the *Debugging Handbook*.

Dump Map Entry

Size: 16 bytes.

Created by: AMDPRREAD and AMDPRLOD.

Updated by: None.

Use: One dump map entry is built for each block of records in the input dump data set. The dump map eliminates searching the data for a requested record.

Offset		Size	Field Name	Description
0	(0)	4	DUMPLINK	Address of next entry in this map.
4	(4)	4	DUMPFADD	Address of the first byte of the first record in the block of records represented by this entry.
8	(8)	4	DUMPLADD	Address of the first byte of the last record in the block of records represented by this entry.
12	(C)	4	DUMPTTR	TTR or block number (tape input) in the input data set of the first record in the block of records represented by this entry.

EDIT Communication Table (AMDPRTAB)

Size: 536 bytes.

Created by: AMDPROOT.

Updated by: AMDPRSCN, AMDPROOT, AMDPRSN2, AMDPRSN3, AMDPRGET, AMDPRREC, AMDPRFLT, AMDPRFRM, AMDPREXT, AMDPRAPP, AMDSYS07.

Use: Communication among AMDPRDMP modules that execute the EDIT function.

Offset		Size		Field Name	Description
0	(0)	4		AFMG	Address of AMDPRFMG message CSECT in AMDPRXED.
4	(4)	4		CURREC	Address of current input record.
8	(8)	12		DEBGFLGS	Debug flags:
8	(8)	1		PTHFLGS1	Flags indicating routine in execution:
		1...	....	ROOT	AMDPROOT.
		.1..	....	SCN	AMDPRSCN.
		..1.	....	GET	AMDPRGET.
		...1	....	CON	AMDPRGET.
		....	1...	HEX	AMDPRCON.
		....	.1..	FLT	AMDPRFLT.
		....	..1.	FRM	AMDPRFRM.
		....	...1	REXT	AMDPREXT.
9	(9)	1		PTHFLGS2	Flags indicating routine in execution: **Bit** **Routine**
				APP	0 AMDPRAPP. 1-7 Reserved.

Offset		Size		Field Name	Description
10	(A)	1		INRFCFGS	Interface flags:
		1...	....	FLMODE	Flush mode.
		.1..	....	TERM	Termination requested.
		..1.	....	SPIE	SPIE routine.
		...1	....	FMT	Formatting of input record requesting by user exit routine.
		....	1...	RET	Indent output (used by AMdPRAPP).
		....	.1..	EXTTRC	External data set being processed.
		....	..1.	EDITSTOP	Stop in progress.
		....	...1	DMDFMT	Unconditional formatting requested by user exit routine.
11	(B)	1		IOFLGS	I/O flags:
		1...	....	GETEOF	End-of-file.
		.xxx	xxxx		Reserved.
12	(C)	8			Reserved.
20	(14)	4		GTFWDPTR	Address of GTF option word, extracted from time stamp record.
24	(18)	8		USEREXIT	User exit name, in EBCDIC.
32	(20)	8		DDNAME	DDNAME keyword value in EBCDIC.
40	(28)	12		STARTIME	Start time value:
40	(28)	8		TIME	Timer units.
48	(30)	2		D	Zeros and year.
50	(32)	2		DAY	Julian day.
52	(34)	12		STOPTIME	Stop time value
52	(34)	8		TIME2	Timer units.
60	(3C)	2		F	Zeros and year.
62	(3E)	2		DAY2	Julian day.
64	(40)	40		JOBNAMES	Maximum of five jobnames in EBCDIC.
104	(68)	20		ASCBADDR	Maximum of five ASCB addresses in hexadecimal.
124	(7C)	1		SIOFLGS	SSCH selectivity flags:
		1...	....	ALLS	EDIT all SSCH records.
		.1..	....	SELS	EDIT selective SSCH records.
		..1.	....	EQUIV	SSCH = IO.
		...1	....	NOEQU	SSCH may not equal IO.
		....	1...	CCWS	Edit CCW records for SSCH events.
		....	.xxx		Reserved.
125	(7D)	3			Reserved.
128	(80)	100		SIODVADS	Maximum of fifty SSCH device addresses.
288	(E4)	1		IOFLGS2	I/O selectivity flags:
		1...	....	ALLI	EDIT all I/O records.
		.1..	....	SELI	EDIT selective I/O records.
		....	1...	CCWI	EDIT CCW records for IO events.
		..xx	.xxx		Reserved.
229	(E5)	3			Reserved.
232	(E8)	100		IODVADS	Maximum of 50 I/O device addresses.
332	(14C)	1		SVCFLGS	SVC selectivity flags:
		1...	....	ALLV	EDIT all SVC records.
		.1..	....	SELV	EDIT selective SVC records.
		..xx	xxxx		Reserved.
333	(14D)	3			Reserved.
336	(150)	32		SVCNUMS	SVC bit string: one bit for each SVC number.
368	(170)	1		USRFLGS	User selectivity flags:
		1...	....	ALLU	EDIT all user records.
		.1..	....	SELU	EDIT selective user records.
		..xx	xxxx	----	Reserved.
369	(171)	3			Reserved.
372	(144)	80		USRNGTAB	Maximum of 20 user EID ranges.
452	(1C4)	1		PIFLGS	PI selectivity flags:
		1...	....	ALLP	EDIT all PI records.
		.1..	....	SELP	EDIT selective PI records.
		..xx	xxxx		Reserved.
453	(1C5)	3	----	Reserved.	
456	(1C8)	32		PICODES	PI bit string: one bit for each PI code.

Offset		Size		Field Name	Description
488	(1E8)	1		GENFLAGS	General flags:
		1...	....	EXT	EDIT external interruption.
		.1..	....	DSP	EDIT dispatcher event records.
		..1.	....	SYS	EDIT GTF comprehensive trace records.
		...1	....	SYSM	EDIT GTF minimal trace records.
		....	1...	RNIO	EDIT RNIO trace records.
		....	.1..	SRM	EDIT SRM trace records.
		....	..1.	RR	EDIT RR trace records.
		....	...1	EOF	End-of-file exit in effect.
489	(IE9)	1		GENFLGS1	General flags:
		1...	....	TS	Time stamp needed.
		.1..	....	EOFINPRO	End of file in progress.
		..1.	....	TSFOUND	Time stamp found.
		...1	....	FIRSTHSW	Hexadecimal dump first time.
		....	1...	SLIPPERM	EDIT SLIP records.
		....	.1..	CPUSEL	CPU selection in effect.
		....	..xx	----	Reserved.
490	(1EA)	3		----	Reserved.
493	(1ED)	3		RECDLL	Record length.
496	(1F0)	8		EXITNM	Name of user exit routine or format appendage currently in control.
504	(1F8)	4		EXITADDR	User exit routine or format appendage entry point address.
508	(1FC)	4		AEIOCT	Address of EDIT I/O control table.
512	(200)	4		PRFMTADD	Address of AMDPRFMT.
516	(204)	4		REENTWKA	Address of workarea.
520	(208)	4		AFRMAD	Address of AMDPRFRM address table.
524	(20C)	2		OFSTEID	Offset of EID in trace record.
526	(20E)	2		OFSTDATA	Offset of data in trace record.
528	(210)	4		ADTSBUF	Address of timestamp record buffer.
532	(214)	9		ESTARTME	Start time save area.
532	(214)	3		ESDAY	Day.
535	(217)	2		ESHR	Hour.
537	(219)	2		ESMIN	Minutes.
539	(21B)	2		ESSEC	Seconds.
541	(21D)	9		ESTOPTME	Stop time save area.
541	(21D)	3		ESPDAY	Day.
544	(220)	2		ESPHR	Hour.
546	(222)	2		ESPMIN	Minutes.
548	(224)	2		ESPSEC	Seconds.
550	(226)	4		CVVTZONE	CVT time-zone value.
554	(22A)	8		----	Reserved.
562	(232)	1		CPU	CPU for select.
563	(233)	1		----	Reserved.
564	(234)	4		ASY07WA	Address of work area for CCW format appendage.

Exit Control Table (ECT)

Size: Variable. The user may use the linkage editor to expand the table.

Created by: The ECT is in SYS1.LINKLIB as CSECT AMDPRECT in load module AMDPRECT.

Updated by: The user updates the ECT entries using the AMASPZAP service aid program. The ECT has at least five available unused entries.

Use: One twenty-byte entry exists for each user-exit module. AMDPRUIM scans the ECT to determine when user-exit modules should receive control. The IHAECTE macro maps an ECT entry.

Offset		Size		Field Name	Description
0	(0)	8		ECTEMODN	Exit module name.
8	(8)	1		EXTEEXIT	Calling flags.
		1...	....	EXTEEXTC	TCB exit.
		.1..	....	ECTEEXAS	ASCB exit.
		..1.	....	ECTEEXFT	FORMAT exit.
		...1	....	ECTEEXPC	PRINT CURRENT/JOBNAME exit.
		....	1...	ECTEEXNU	PRINT NUCLEUS exit.
		....	.1..	ECTEXHD	Header exit.
		....	..xx	----	Reserved.
9	(9)		1		ECTEIFAT Interface attributes.
		1...	....	ECTEIFSA	The PRDMP access storage service must treat the virtual address input parameter passed to it in register 0 as a 24-bit value when it is invoked by this module for locations in the dump. Although this module limits its requests for virtual storage to locations below 16M, it does not guarantee that the high order byte of register 0 is zero.
		.xxx	xxxx	----	Reserved.
10	(A)	2		ECTERSRV	Reserved.
12	(C)	8		ECTEVERB	User verb name.

Path Descriptor Element

Size: 4 bytes.

Created by: AMDPRCTL.

Updated by: AMDPRCTL.

Use: One for each nonresident service and executor module; describes the module for loading by AMDPRSEG.

Offset		Size		Field Name	Description
0	(0)	1		Flags	Flags:
		0...	....		Service module.
		1...	....		Executor module.
		.xxx	xxxx		Reserved
1	(1)	3			Module ID: the unique last three characters of the module name.

Print Control Block (PCB)

Size: 16 bytes.

Created by: AMDPRPCB.

Updated by: None.

Use: Contains the starting and ending addresses in the dump data set of storage areas to be printed by AMDPRDMP.

Offset		Size		Field Name	Description
0	(0)	4		PCBLINK	Address of next PCB.
4	(4)	4		PCBSTART	Starting address of storage to be printed.
8	(8)	4		PCBSTOP	Ending address of storage to be printed.
12	(C)	2		PCBASID	ID of address space to be printed.
14	(3)	1			Flags:
		1...	....		Virtual storage PCB.
		.1..	....		Real storage PCB.
		..1.	....		Change PCBSTART value to zero.
		...1	....		Update TOPICHDR field in COMMON.
		....	xxxx		Reserved.

Print Dump Index Description Table (AMDMNDXT)

Size: Each entry is 44 bytes; the total length depends on total number of description entries. For each entry, there is a 4-byte table entry description word (EDW), and a 40-byte description data area.

EDW — 4 bytes.

Data — 40 bytes per each description entry.

Created by: AMDPRCOM.

Use: AMDMNDXT is the data area that contains the print dump index description table and its entry constants that are in the form of assembler EQU instructions.

EDW format for each entry:

Offset		Size		Field Name	Description
0	(0)	1		----	Reserved.
1	(1)	1	----		Insertion data is needed.
		1...	....		
2	(2)	1		----	The offset in the description data where data is inserted.
3	(3)	1		----	Length of the insertion data.

The description table contains the following 40 byte long entries. The 4 byte EDW precedes each description entry.

```
'PRINT DUMP ABSTRACT INFORMATION . . . . . . . . . . '
'COMMUNICATION VECTOR TABLE (CVT). . . . . . . . . . '
'NUCLEUS STORAGE . . . . . . . . . . . . . . . . . . '
'DAT-ON NUCLEUS  . . . . . . . . . . . . . . . . . . '
'DAT-OFF NUCLEUS . . . . . . . . . . . . . . . . . . '
'SYSTEM QUEUE AREA (SQA) STORAGE . . . . . . . . . . '
'COMMON SYSTEM AREA (CSA) STORAGE  . . . . . . . . . '
'LINK PACK AREA MAP  . . . . . . . . . . . . . . . . '
'LPDE SORTED BY NAME MAP . . . . . . . . . . . . . . '
'LPDE SORTED BY EPA MAP  . . . . . . . . . . . . . . '
'LPA ACTIVE QUEUE MAP  . . . . . . . . . . . . . . . '
'SYSTEM SUMMARY  . . . . . . . . . . . . . . . . . . '
'DUMP ADDRESS RANGES SUMMARY . . . . . . . . . . . . '
'ACTIVE CPU LIST SUMMARY . . . . . . . . . . . . . . '
'SCHEDULED SERVICES (SRB) SUMMARY  . . . . . . . . . '
'JOB SUMMARY . . . . . . . . . . . . . . . . . . . . '
'PROBLEM LIST SUMMARY  . . . . . . . . . . . . . . . '
'PRINT DUMP FORMAT . . . . . . . . . . . . . . . . . '
'SCHEDULED SERVICES (SRB)  . . . . . . . . . . . . . '
'PRINT DUMP EDIT . . . . . . . . . . . . . . . . . . '
'VIRTUAL STORAGE PRINT . . . . . . . . . . . . . . . '
'REAL STORAGE PRINT  . . . . . . . . . . . . . . . . '
'PRIVATE STORAGE FOR ASID XXXX . . . . . . . . . . . '
'NON-EXTENDED COMMON STORAGE PRINT . . . . . . . . . '
'NUCLEUS STORAGE PRINT . . . . . . . . . . . . . . . '
'EXTENDED COMMON STORAGE PRINT . . . . . . . . . . . '
'CURRENT STORAGE PRINT . . . . . . . . . . . . . . . '
'JOBNAME PRINT . . . . . . . . . . . . . . . . . . . '
'ADDRESS SPACE XXXX CONTROL BLOCKS . . . . . . . . . '
'TASK CONTROL BLOCK (TCB)  . . . . . . . . . . . . . '
'TASK CONTROL BLOCK SUMMARY  . . . . . . . . . . . . '
'REQUEST BLOCKS (RB) . . . . . . . . . . . . . . . . '
'EXTENDED STATUS BLOCKS (XSB)  . . . . . . . . . . . '
'PROGRAM CALL LINK STACK (STKE)  . . . . . . . . . . '
'LOAD LIST . . . . . . . . . . . . . . . . . . . . . '
'JOB PACK QUEUE (JPQ)  . . . . . . . . . . . . . . . '
'IOS DATA EXTENT BLOCKS (DEB)  . . . . . . . . . . . '
'TASK INPUT/OUTPUT TABLE (TIOT)  . . . . . . . . . . '
'CROSS MEMORY INFORMATION  . . . . . . . . . . . . . '
'IOS DATA AREAS  . . . . . . . . . . . . . . . . . . '
'RTM DATA AREAS  . . . . . . . . . . . . . . . . . . '
'DATA MANAGEMENT DATA AREAS  . . . . . . . . . . . . '
'PROCESSOR RELATED DATA AREAS (CPUDATA)  . . . . . . '
'CONSOLE INITIATED LOOP TRACE RECORDS  . . . . . . . '
```

The Index Table Entry Code Equates List:

Field Name	Equate	Description
INDXABS	1	AMDPRABS, PRINT DUMP ABSTRACT
INDXCVT	2	AMDPRCVT, CVT MA
INDXNUC	3	AMDPRNUC, NUCLEUS PRINT
INDXDTN	4	AMDPRNUC, DAT-ON NUCLEUS PRINT
INDXDTF	5	AMDPRNUC, DAT-OFF NUCLEUS PRINT
INDXSQA	6	AMDPRNUC, SQA PRINT
INDXCSA	7	AMDPRNUC, CSA PRINT
INDXLPA	8	AMDPRLPA, LINK PACK AREA MAP INDEX TITLE
INDXLPA1	9	AMDPRLPA, LPDE SORTED BY EPA MAP
INDXLPA2	10	AMDPRLPA, LPDE SORTED BY NAME MAP
INDXLPA3	11	AMDPRLPA, LPA ACTIVE QUEUE MAP
INDXSUM	12	BLSQSUM4, SYSTEM SUMMARY

Field Name	Equate	Description
INDXSUM1	13	BLSQSUM4, DUMP ADDRESS RANGES SUMMARY
INDXSUM2	14	BLSQSUM4, CPU LIST SUMMARY
INDXSUM3	15	BLSQSUM4, SCHEDULED SERVICES SUMMARY
INDXSUM4	16	BLSQSUM4, JOB SUMMARY
INDXSUM5	17	BLSQSUM2, PROBLEM LIST SUMMARY
INDXFXT	18	BLSQSUM2, PRINT DUMP FORMAT
INDXFSRB	19	BLSQSUM2, SCHEDULED SERVICES (SRB)
INDXEDIT	20	AMDPRFRM, PRINT DUMP EDIT
INDXVSPR	21	AMDPRPMS, VIRTUAL STORAGE PRINT
INDXRSPR	22	AMDPRPMS, REAL STORAGE PRINT
INDXPAID	23	AMDPRPMS, PRIVATE STORAGE FOR ASID XXXX AMDPRPMS, PRIVATE STORAGE FOR ASID XXXX AMDPRPMS, PRIVATE STORAGE FOR ASID XXXX
INDXPNEC	24	AMDPRPMS, NON-EXTENDED COMMON STORAGE PRINT
INDXPNUC	25	AMDPRPMS, NUCLEUS STORAGE PRINT
INDXPECS	26	AMDPRPMS, EXTENDED COMMON STORAGE PRINT
INDXPCR	27	AMDPRPCR, PRINT CURRENT
INDXPJBN	28	AMDPRPJB, PRINT JOBNAME
INDXASCB	29	BLSQSUM2, ADDR SPACE XXXX CONTROL BLOCKS
INDXFTCB	30	BLSQSUM2, TASK CONTROL BLOCKS (TCB) FORMAT
INDXTSUM	31	BLSQSUM2, TCB SUMMARY
INDXFRB	32	BLSQSUM2, REQUEST BLOCK (RB)
INDXFXSB	33	BLSQSUM2, EXTENDED STATUS BLOCK (XSB)
INDXSTKE	34	BLSQSUM2, PROGRAM CALL LINK STACK (STKE)
INDXFLL	35	BLSQSUM2, LOAD LIST
INDXFJPQ	36	BLSQSUM2, JOB PACK QUEUE
INDXFDEB	37	BLSQSUM2, IOS DATA EXTENT BLOCKS (DEB)
INDXTIOT	38	BLSQSUM2, TASK INPUT/OUTPUT TABLE (TIOT)
INDXFXM	39	AMDPRXMT, CROSS MEMORY INFORMATION
INDXIOSD	40	IECIOFMT, IOS DATA AREAS
INDXRTMD	41	IEAVTFMT, RTM DATA AREAS
INDXDMDA	42	IECDAFMT, DATA MANAGEMENT DATA AREAS
INDXCPUD	43	AMDPRGCD, PROCESSOR RELATED DATA AREAS
INDXLRF	44	AMDPRLRF, CONSOLE INITIATED LOOP TRACE RECORDS
INDXETIT	254	AMDPRFRM, INITIALIZATION THROUGH THE EDIT VERB
INDXATIT	255	AMDPRABS, INITIALIZATION DURING ABSTRACT PROCESSING

TCBLIST Entry

Size: 8 bytes.

Created by: CSECT AMDPRTSV in AMDPRCOM.

Updated by: CSECT AMDPRTSV in AMDPRCOM.

Use: Describes selected TCBs for use by subsequent routines.

Offset		Size	Field Name	Description
0	(0)	4		Address of next TCBLIST entry.
4	(4)	4		Address of associated TCB.

Section 5: Diagnostic Aids

This section contains the following information to aid the reader in diagnosing AMDPRDMP errors:

- AMDPRDMP register usage.
- Return codes issued by AMDPRDMP modules.

Note: Refer to *System Messages* for message text and detecting, issuing, and containing modules.

AMDPRDMP Register Conventions

Symbol	Register	Usage
PREG	1	Parameter register.
R2	2	Parameter register in following CSECTs: ● CSECT AMDPRMSG in AMDPRCOM — Length of message to be written. ● CSECT AMDPRTSV in AMDPRCOM — Contents depend on routine: 1. TCBQSAVE — TCB address. 2. TCBRTRV — TCBLIST position number. 3. TCBREMV — Address of TCB to be removed or 0 to indicate that all TCBs are to be dequeued.
BUFREG	6	Base of DSECT for output line (OUTBUFM).
R7	7	TCB address passed to TIOT element enqueue routine in AMDPRFUB.
R9	9	● During EDIT processing, the address of the EDIT communication table (AMDPRTAB). ● During processing other than EDIT, R9 is the linkage register for internal subroutines.
BASE1	11	Base register for all modules.
COMBASE	12	Address of AMDPRDMP common communication area (COMMON).
R13	13	Address of standard register save area.
RETREG	14	Linkage register for CSECT to CSECT subroutine calls.
R15	15	Entry point register for CSECT to CSECT subroutine calls.

AMDPRDMP Return Codes

Module	CSECT	Return Code	Meaning
AMDPRABS	AMDPRABS	0	Header record available, valid dump type specified.
		8	Invalid dump type specified.
		12	Header record not available.
AMDPRAPP	AMDPRAPP	None	
AMDPRCMC	AMDPRCMC	None	
AMDPRCOM	AMDPRCOM	None	
	AMDPRGFR	None	
	AMDPRMSG	None	
	AMDPRNDX	None	
	AMDPRSTP	None	
	AMDPRSYN	None	
	AMDPRTSV	0	Successful completion of TCBLIST enqueue, dequeue, or retrieve operation.
	AMDPRWTR	None	

Module	CSECT	Return Code	Meaning
AMDPRCTL	AMDPRCTL	None	
	AMDPREND	4	AMDPRDMP termination in progress.
	AMDPRMSC	0	Control statement has been completely processed.
		4	Error discovered during control statement processing.
	AMDPRXIT	None	
AMDPRCVT	AMDPRCVT	0	CVT formatting completed.
		4	Storage not available.
AMDPRDPS	AMDPRDPS	None	
	AMDPRPCB	0	A PCB has been dequeued and the information placed in the caller's work area.
		4	PCB queue is empty.
AMDPREAD	AMDPREAD	None	
AMDPRECT	AMDPRECT		Not applicable.
AMDPREXT	AMDPREXT	None	
AMDPRFAR	AMDPRFAR	None	
AMDPRFLT	AMDPRFLT	None	
AMDPRFMG	AMDPRFMG	Not applicable	
AMDPRFMT	AMDPRFMT	0	All formatting completed.
		4	Formatting incomplete.
AMDPRFOR	AMDPRFOR	None	
AMDPRFRM	AMDPRFRM	None	
AMDPRFSR	AMDPRFDB	None	
	AMDPRFIO	None	
	AMDPRFLD	None	
	AMDPRFRB	None	
	AMDPRFSC	None	
	AMDPRFTC	0	Formatted TCB represents task that was active in dumped system.
		4	Formatted TCB represents task that was terminated in dumped system.
AMDPRFUB	AMDPRFUB	None	
AMDPRFXT	AMDPRFXT	None	
AMDPRGCD	AMDPRGCD	None	
AMDPRGET	AMDPRGET	None	
AMDPRGSA	AMDPRGSA	None	
	AMDPRGFT	None	
	AMDPRGNX	None	
AMDPRJNA	AMDPRJNA	0	JOBNAME valid, HOME ASID and ASCB address.
		4	ABDPL unavailable.
		8	CVT unavailable.
		12	ASVT unavailable.
		16	No matching jobname.
AMDPRLOD	AMDPRLOD	0	Loading of SYSUT1 work data set is complete;dump formatting may begin.
		4	The input data set cannot be positioned to the requested information.
		8	Terminate AMDPRDMP; load mode requested and EOF encountered.
AMDPRLPA	AMDPRLPA	None	
AMDPRLRF	AMDPRLRF	None	
AMDPRMST	AMDPRMST	None	
AMDPRNUC	AMDPRNUC	None	
AMDPROOT	AMDPROOT	None	
AMDPRPCR	AMDPRPCR	None	
AMDPROOT	AMDPROOT	None	
AMDPRPJB	AMDPRPJB	None	
AMDPRPMG	AMDPRPMG	Not applicable	

Module	CSECT	Return Code	Meaning
AMDPRPMS	AMDPRPMS	None	
	AMDPRPMX	None	
AMDPRRDC	AMDPRRDC	4	Storage is not available for buffers or there is no valid input data set.
AMDPRREC	AMDPRCON	None	
	AMDPRHEX	8	Processing is complete.
	AMDPRTME	None	
AMDPRSCN	AMDPRSCN	0	EDIT control statement syntax is correct and EDIT processing may continue.
		4	Discontinue EDIT processing for one of the following reasons: ● AMDPRDMP control statements are being entered from the SYSIN data set, the EDIT control statement indicates that trace data is to be edited from a dump data set, and the AMDPRDMP flush mode switch is on. ● AMDPRDMP control statements are being entered from the SYSIN data set and an error has been discovered in the EDIT control statement.
AMDPRSEG	AMDPRSEG	0	Successful completion of module loading, or TOD conversion.
		4	User format appendage greater than 10K and could not be loaded, or TOD value was zero.
		8	Module not found by BLDL macro instruction. Module name is given in message AMD177I.
		12	I/O error during execution of BLDL macro instruction.
		16	AMDPRSEG attempted to delete a module that was not currently in virtual storage.
AMDPRSLI	AMDPRSLI	None	
AMDPRSMG	AMDPRSMG	None	
AMDPRSN2	AMDPRSN2	0	A syntax error has been discovered by a keyword subroutine; flush mode processing must begin.
		4	Keyword processing is complete; AMDPRSCN must determine whether the end of the current control statement has been reached.
		8	AMDPRSCN is requested to determine whether unmatched parentheses are in the EDIT control statement.
		12	AMDPRSN2 must issue the invalid parenthesis error message.
		16	Control statement scan is to continue; AMDPRSCN will attempt to isolate the next EDIT keyword. AMDPRSN3 AMDPRSN3 0 EDIT control statement syntax is correct.
		4	Error in EDIT control statement.
AMDPRSUM	AMDPRSUM	None	
AMDPRSUM	AMDPRSUM	0	Dump data read was successful.
		4	Requested data could not be read.
AMDSYS00	AMDSYS00	0	First output line has been formatted: AMDPRAPP should print this line and return to AMDSYS00.
		4	Second output line has been formatted; formatting for this trace record is complete. AMDPRAPP should print the line and complete processing for this record.
		8	Error encountered in formatting the trace record, or selective editing was in effect and this record is not to be edited.
		12	Record should be dumped in hexadecimal.

Module	CSECT	Return Code	Meaning
AMDSYS01	AMDSYS01	0	First output line has been formatted: AMDPRAPP should print the line and return to AMDSYS01.
		4	Second output line has been formatted; formatting for the current record is complete. AMDPRAPP should print the formatted line
		8	Error encountered in formatting the trace record, or selective editing is in effect and this trace record is not to be formatted.
		12	Record should be dumped in hexadecimal.
AMDSYS02	AMDSYS02	0	First output line has been formatted: AMDPRAPP should print the line and return to AMDSYS02.
		4	Second output line has been formatted; formatting for this record is complete. AMDPRAPP should print the line.
		8	Error encountered in formatting the record, or selective editing is in effect and this record is not to be formatted.
		12	Record should be dumped in hexadecimal.
AMDSYS03	AMDSYS03	4	Current trace record has been formatted; AMDPRAPP should print the record.
		8	Error encountered in formatting the record, or selective editing is in effect and this record is not to be formatted.
		12	Record should be dumped in hexadecimal.
AMDSYS04	AMDSYS04	0	First output line has been formatted: AMDPRAPP should print the line and return to AMDSYS04.
		4	Second output line has been formatted; formatting for this record is complete. AMDPRAPP should print the line.
		8	Error encountered in formatting the record, or selective editing is in effect and this record is not to be formatted.
		12	Record should be dumped in hexadecimal.
AMDSYS05	AMDSYS05	0	First output line has been formatted; AMDPRAPP should print the line and return to AMDSYS05.
		4	Second output line has been formatted; formatting for this record is complete. AMDPRAPP should print the line.
		8	Error encountered in formatting the record, or selective editing is in effect and this record is not to be formatted.
		12	Record should be dumped in hexadecimal.
AMDSYS06	AMDSYS06	0	First output line has been formatted: AMDPRAPP should print the line and return to AMDSYS06.
		4	Second output line has been formatted; formatting for this record is complete. AMDPRAPP should print the line.
		8	Error encountered in formatting the record, or selective editing is in effect and this record is not to be formatted.
AMDSYS07	AMDSYS07	0	The return code is zero when any output line except the final output line has been formatted. AMDPRAPP should print the line and return to AMDSYS07.
		4	Final output line has been formatted; formatting for this record is complete. AMDPRAPP should print the line.
		8	Error encountered in formatting the record, or selective editing is in effect and this record is not be formatted.
		12	Record should be dumped in hexadecimal.

Section 6: AMDPRDMP Macro Instructions

AMDDATA

The AMDDATA macro is a mapping macro for the various input record formats accepted by AMDPRDMP. AMDPRDMP modules use this macro instruction for symbolic references to fields within the input. AMDDATA is also available to the AMDSADMP and SVC dump programs to define their output records (which are later input to AMDPRDMP).

AMDPCBPL

The AMDPCBPL macro is a mapping macro for the PCB to be used as input to the AMDPRPCB routine in module AMDPRDPS.

AMDMNDXT

The AMDMNDXT macro generates the index descriptor table. All AMDPRDMP modules that have predefined index entries use this macro.

BRPRTMSG

The BRPRTMSG macro generates linkage to CSECT AMDPRMSG in AMDPRCOM in order to write a message in SYSPRINT output data set. The BRPRTMSG macro instruction has two parameters: the first is the address of the message to be printed and the second is the length of the message.

BRREAD

The BRREAD macro generates linkage to AMDPRRDC to perform one of three functions, depending on the keyword specified as the first or second parameter in the macro instruction:

- ADJUST — causes AMDPRRDC to free buffers and the buffer map.
- INIT — causes AMDPRRDC to load AMDPRLOD to write the input data set into SYSUT1 or SYSUT2 (if provided in JCL); to initialize buffers and a buffer map; and, for each new dump, to call AMDPRMST to initialize COMMON, and to build ASCBMAP.
- DATA — causes AMDPRRDC to locate the input record that contains the requested data and return the address of the data in the buffer to the caller. For details of the DATA function, refer to the description of reading input data in the "Program Organization" section.
- LENGTH — is meaningful only as an option of the DATA function. LENGTH causes AMDPRRDC to return the specified amount of data (up to 4K maximum) to the issuer. The default is a length of zero and indicates that the length of the data depends on the boundary of the specified address.

BRWRITE

The BRWRITE macro generates the linkage to CSECT AMDPRWTR in AMDPRCOM in order to write a line in the PRINTER output data set. The information to be written must be in the buffer pointed to by the CURBUF field of COMMON. The BRWRITE macro instruction contains two parameters. The first is either 1, 2, 3 or SKIP to indicate the number of lines to skip after the written line. The second parameter is a keyword, either IMM or AFT (or blank), which indicates whether to space immediately or after writing the CURBUF field.

COMMON

The COMMON macro is the mapping macro for the common communication area (COMMON). It is used by all AMDPRDMP modules to generate the COMMON DSECT. For a detailed description of COMMON, see the "Data Areas" section of this chapter.

EQUATES

The EQUATES macro instruction generates a list of EQU instructions used by all AMDPRDMP modules.

FMTPTRN

The FMTPTRN macro instruction generates a parameter list that is passed to CSECT AMDPRGFR in AMDPRCOM in order to format a line of output. Each FMTPTRN macro instruction must contain the attributes of the data, the length of the label to be assigned, the length of the data, the offset of the label in the output line, the offset of the data, and, optionally, the address of the label string and the address of the data.

HEXCNVT

The HEXCNVT macro instruction generates the code to convert a value from internal binary to printable hexadecimal.

IMDMEDIT

The IMDMEDIT macro instruction generates a list of symbolic names with equated hexadecimal event identifiers (EIDs) for events that are traced by GTF. IBM components and users that write EDIT user exit routines use the IMDMEDIT macro instruction to reference the EID field in a trace record built by GTF.

The IMDMEDIT mapping macro instruction is as follows:

```
IMDMSSM      EQU   0            OS SSM FOR COMPATIBILITY
IMDMSSM1     EQU   0            SSM INTERRUPT
IMDMPIPG     EQU   0            PAGE FAULT PROGRAM INTERRUPT
IMDMDSP1     EQU   X'0001'      DISPATCHER
IEADISP1     EQU   IMDMDSP1     DISPATCHER
IMDMDSP2     EQU   X'0002'      DISPATCHER
```

IEADISP2	EQU	IMDMDSP2	DISPATCHER
IMDMDSP	EQU	X'0003'	DISPATCHER
IMDMDSP3	EQU	X'0003'	DISPATCHER
IEADISP3	EQU	IMDMDSP3	DISPATCHER
IMDMDSP4	EQU	X'0004'	SVC EXIT PROLOG DISPATCH
IEADISP4	EQU	IMDMDSP4	EXIT PROLOG DISPATCH
IMDMSVC	EQU	X'1000'	SVC INTERRUPT
IEASVCH	EQU	IMDMSVC	SVC INTERRUPT
IMDMPCI	EQU	X'2100'	PCI I/O INTERRUPT
IECPCI	EQU	IMDMPCI	PCI I/O INTERRUPT
IMDMSRM	EQU	X'4001'	SRM
IRASRM	EQU	IMDMSRM	SRM
IMDMSTAE	EQU	X'4002'	RTM
IEASTAE	EQU	IMDMSTAE	RTM
IMDMFRR	EQU	X'4003'	RTM
IEAFRR	EQU	IMDMFRR	RTM
IMDMSLSD	EQU	X'4004'	RTM/SLIP STANDARD RECORD
IEAVSLSD	EQU	IMDMSLSD	RTM/SLIP STANDARD RECORD
IMDMSLSU	EQU	X'4005'	RTM/SLIP STANDARD + USER RECORD
IEAVSLSU	EQU	IMDMSLSU	RTM/SLIP STANDARD + USER RECORD
IMDMSLUR	EQU	X'4006'	RTM/SLIP USER RECORD
IEAVSLUR	EQU	IMDMSLUR	RTM/SLIP USER RECORD
IMDMSIO	EQU	X'5100'	SIO OPERATION
IECSIO	EQU	IMDMSIO	SIO OPERATION
IMDMEOS	EQU	X'5101'	IOS
IECEOS	EQU	IMDMEOS	IOS
IMDMCSCH	EQU	X'5102'	CLEAR SUBCHANNEL GTF RECORD
IECCSCH	EQU	IMDMCSCH	CLEAR SUBCHANNEL GTF RECORD
IMDMHSCH	EQU	X'5103'	HALT SUBCHANNEL GTF RECORD
IECHSCH	EQU	IMDMHSCH	HALT SUBCHANNEL GTF RECORD
IMDMMSCH	EQU	X'5104'	MODIFY SUBCHANNEL GTF RECORD
IECMSCH	EQU	IMDMMSCH	MODIFY SUBCHANNEL GTF RECORD
IMDMSSCH	EQU	X'5105'	START SUBCHANNEL GTF RECORD
IECSSCH	EQU	IMDMMSCH	START SUBCHANNEL GTF RECORD
IMDMRSCH	EQU	X'5106'	RESUME SUBCHANNEL GTF RECORD
IECRSCH	EQU	IMDMRSCH	RESUME SUBCHANNEL GTF RECORD
IMDMIO2	EQU	X'5200'	I/O INTERRUPT
IECIO2	EQU	IMDMIO2	I/O INTERRUPT
IMDMIO1	EQU	X'5201'	I/O INTERRUPT
IECIO1	EQU	IMDMIO1	I/O INTERRUPT
IMDMPI	EQU	X'6101'	PROGRAM INTERRUPT
IEAPINT	EQU	IMDMPI	PROGRAM INTERRUPT
IMDMTINT	EQU	X'6200	PFLIH
IEATINT	EQU	IMDMTINT	PFLIH
IMDMEXT	EQU	X'6201'	EXTERNAL INTERRUPT
IEAEINT	EQU	IMDMEXT	EXTERNAL INTERRUPT
IMDMTP1	EQU	X'8100'	TPIOS
ISPTPIO1	EQU	IMDMTP1	TPIOS
IMDMTP2	EQU	X'8200'	TPIOS
ISPTPIO2	EQU	IMDMTP2	TPIOS
IMDGPD01	EQU	X'EFAF'	RESERVED
IMDGPD02	EQU	X'EFB0'	RESERVED
IMDGPD03	EQU	X'EFB1'	RESERVED
IMDGPD04	EQU	X'EFB2'	RESERVED
IMDGPD05	EQU	X'EFB3'	RESERVED
IMDGPD06	EQU	X'EFB4'	RESERVED
IMDGPD07	EQU	X'EFB5'	RESERVED
IMDGPD08	EQU	X'EFB6'	RESERVED
IMDGPD09	EQU	X'EFB7'	RESERVED
IMDGPD10	EQU	X'EFB8'	RESERVED
IMDGPD11	EQU	X'EFB9'	RESERVED
IMDGPD12	EQU	X'EFBA'	RESERVED
IMDGPD13	EQU	X'EFBB'	RESERVED
IMDGPD14	EQU	X'EFBC'	RESERVED
IMDGPD15	EQU	X'EFBD'	RESERVED
IMDGPD16	EQU	X'EFBE'	RESERVED
IMDGPD17	EQU	X'EFBF'	RESERVED
IMDGPD18	EQU	X'EFC0'	RESERVED

IMDGPD19	EQU	X'EFC1'	RESERVED
IMDGPD20	EQU	X'EFC2'	RESERVED
IMDGPD21	EQU	X'EFC3'	RESERVED
IMDGPD22	EQU	X'EFC4'	RESERVED
IMDGPD23	EQU	X'EFC5'	RESERVED
IMDGPD24	EQU	X'EFC6'	RESERVED
IMDGPD25	EQU	X'EFC7'	RESERVED
IMDGPD25	EQU	X'EFC8'	RESERVED
IMDGPD27	EQU	X'EFC9'	RESERVED
IMDGPD28	EQU	X'EFCA'	RESERVED
IMDGPD29	EQU	X'EFCB'	RESERVED
IMDGPD30	EQU	X'EFCC'	RESERVED
IMDGPD31	EQU	X'EFCD'	RESERVED
IMDGPD32	EQU	X'EFCE'	RESERVED
IMDGPD33	EQU	X'EFCF'	RESERVED
IMDGPD34	EQU	X'EFD0'	RESERVED
IMDGPD35	EQU	X'EFD1'	RESERVED
IMDGPD36	EQU	X'EFD2'	RESERVED
IMDGPD37	EQU	X'EFD3'	RESERVED
IMDGPD38	EQU	X'EFD4'	RESERVED
IMDGPD39	EQU	X'EFD5'	RESERVED
IMDGPD40	EQU	X'EFD6'	RESERVED
IMDGPD41	EQU	X'EFD7'	RESERVED
IMDGPD42	EQU	X'EFD8'	RESERVED
IMDGPD43	EQU	X'EFD9'	RESERVED
IMDGPD44	EQU	X'EFDA'	RESERVED
IMDGPD45	EQU	X'EFDB'	RESERVED
IMDGPD46	EQU	X'EFDC'	RESERVED
IMDGPD47	EQU	X'EFDD'	RESERVED
IMDGPD48	EQU	X'EFDE'	RESERVED
IMDGPD49	EQU	X'EFDF'	RESERVED
IMDGPD50	EQU	X'EFE0'	RESERVED
ISTVIEIE	EQU	X'EFE1'	ACF/VTAM INTERNAL TRACE
ISTTHEID	EQU	X'EFE2'	TSO/VTAM TGET/TPUT TRACE
ISTTREID	EQU	X'EFE3'	VTAM RESERVED
ISTTDEID	EQU	X'EFE4'	ACF/VTAM NCP LINE TYPE TRACE
ISTTPEID	EQU	X'EFEF'	ACF/VTAM USER BUFFER CONTENTS TRACE
ISTRPEID	EQU	X'EFF0'	ACF/VTAM SMS (BUFFER USE) TRACE
ISTCLEID	EQU	X'EFF1'	ACF/VTAM COMPONENT BUFFER CONTENTS TRACE
ISTLNEID	EQU	X'EFF2'	ACF/VTAM NCP LINE OR TG TRACE
IGGSP002	EQU	X'EFF3'	SAM/PAM/DAM
IGGSP008	EQU	X'EFF4'	SAM/PAM/DAM
IDAAM01	EQU	X'EFF5'	VSAM
IGGSP112	EQU	X'EFF6'	SAM/PAM/DAM
IGGSP215	EQU	X'EFF7'	SAM/PAM/DAM
IGGSP119	EQU	X'EFF8'	SAM/PAM/DAM
IGGSP235	EQU	X'EFF9'	SAM/PAM/DAM
IGGSP239	EQU	X'EFFA'	SAM/PAM/DAM
IGGSP145	EQU	X'EFFB'	SAM/PAM/DAM
IGGSP251	EQU	X'EFFC'	SAM/PAM/DAM
IGGSP451	EQU	X'EFFD'	SAM/PAM/DAM
IGGSP169	EQU	X'EFFE'	SAM/PAM/DAM
IMDMDMA1	EQU	X'EFFF'	OPEN/CLOSE/EOV
IMDMSVCD	EQU	X'F100'	SVCDUMP
IEASVCD	EQU	IMDMSVCD	SVCDUMP
IEAABOF	EQU	X'F200'	ABEND/SNAP-OUT-NOT USED BY EDIT

OUTBUFM

The OUTBUFM macro instruction is the mapping macro for an output line in the PRINTER data set. AMDPRDMP modules use this macro instruction to position data in the line.

SYNEPS

The SYNEPS macro instruction is the mapping macro instruction for the entry points for CSECT AMDPRSYN in AMDPRCOM.

Chapter 4. Stand-Alone Dump (AMDSADMP)

Section 1: Introduction

AMDSADMP Macro Instruction

The stand-alone dump program is supplied to you as a macro definition, AMDSADMP, in the SYS1.MACLIB system library. You code the AMDSADMP macro instruction, using the keyword operands to tailor the dump program to the installation. For a description of the AMDSADMP macro instruction and its keywords, refer to *Service Aids*.

After coding the macro instruction, you put the SADMP program onto the residence volume in ready-to-load form, using either of two methods. In two-stage generation, first you assemble the AMDSADMP macro in order to produce stage-two JCL. Then you execute this job in order to initialize the SADMP residence volume. In one-step generation, you code the same AMDSADMP macro as an input control statement for the AMDSAOSG program, and then execute AMDSAOSG.

In either case, SADMP residence volume initialization consists of three phases:

1. The assembly of the AMDSADM2 macro produces the SADMP real storage dump program AMDSARDM, which dumps real storage, and the SADMP common communication table AMDSACCT, an internal control block.

2. The SADMP build module AMDSABLD puts the output from phase one onto the residence volume in ready-to-load form. AMDSABLD relocates the SADMP IPL program AMDSAIPL and the SADMP virtual storage dump program AMDSAPGE, and puts them onto the residence volume.

3. If the residence volume is a direct access device, the device utility ICKDSF is invoked to put SADMP's IPL text onto the device's IPL track (cylinder 0, track 0).

The SADMP program has four basic variations:

- High-speed, residing on a direct access device, with output directed to a tape volume.

- High-speed, residing on a tape device, with output directed to a tape volume.

- Low-speed, residing on a direct access device, with output directed to a tape volume.
- Low-speed, residing on a direct access device, with output directed to a printer.

High-Speed Dump Program

The high-speed version of SADMP dumps all real storage and areas of virtual storage not backed by real storage. The output is unformatted and intended for later machine processing by AMDPRDMP. The output includes:

- A dump title.
- The processor store-status information for each processor.
- Real storage from address 0 to the top of real storage in real address sequence (some blocks may be missing because they are offline or are backing the hardware system area).
- Instruction trace data created by the console-initiated loop recording.
- Selected virtual storage areas.
- A log of console messages written during the dumping of MVS virtual storage.

If SADMP detected an internal error, the output might also include one or more SADMP self-dumps. SADMP dumps the instruction trace data and paged-out virtual storage only when SADMP is able to access virtual storage.

Notes:

1. *Stand-alone dump uses only online devices. When dumping to or from devices that have both real and virtual addresses, specify only the real address to SADMP. SADMP must reside on an online storage device.*
2. *You cannot direct SADMP output to its residence volume.*

Low-Speed Dump Program

The low-speed version of SADMP produces formatted output containing a dump title specified at dump execution time, processor related data for each available processor, and a dump of user-selected areas of real storage.

Section 2: Method of Operation

This section describes how the stand-alone dump program produces the dumps. As shown in SADMP Figure 4-1, the description begins with an overview, then describes the following stages of the overall processing:

- Specification stage
- Initialization stage
- Dump execution stage

Note: SADMP also formats the console message log under IPCS and PRDMP.

Reading Method of Operation Diagrams

Method of operation diagrams are arranged in an input-processing-output layout: the left side of the diagram contains the data that serves as input to the processing steps in the center of the diagram, and the right side contains the data that is output from the processing steps. Each processing step is numbered; the number corresponds to the verbal description of the step in the extended description. While the processing step in the diagram is in general terms, the corresponding text is a specific description that includes a cross-reference to the code for the processing.

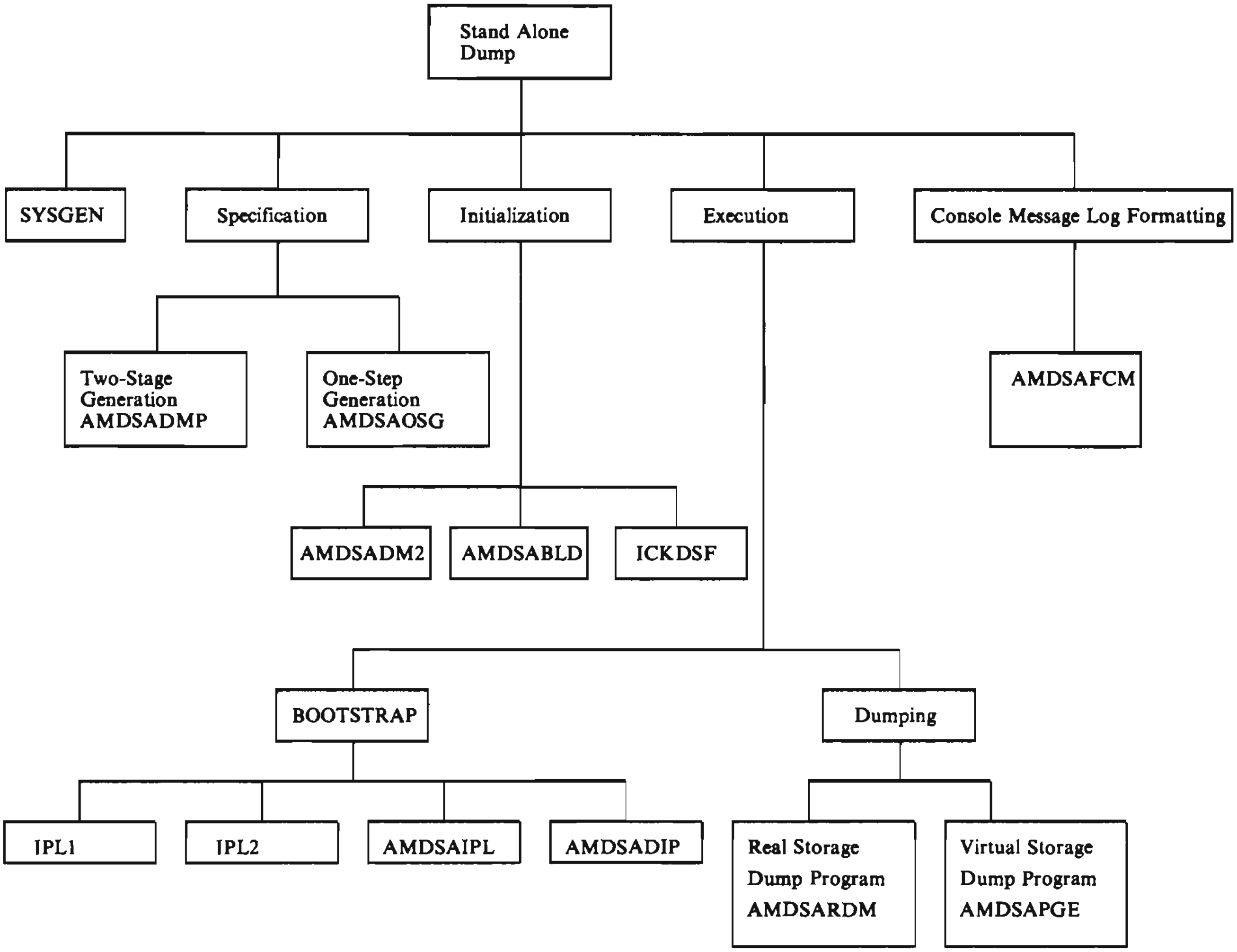


Figure 4-1 (Part 1 of 2). Key to Method of Operation Diagrams for the Stand-Alone Dump Program

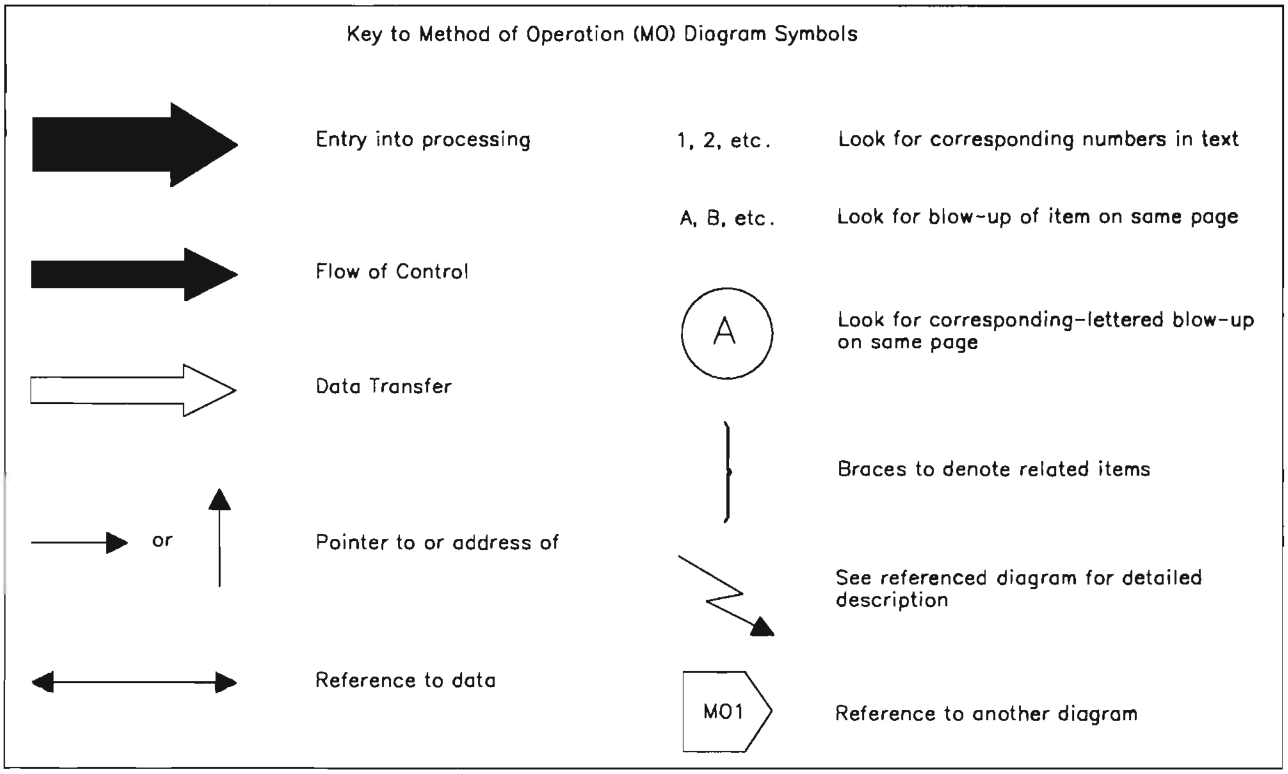


Figure 4-1 (Part 2 of 2). Key to Method of Operation Diagrams for the Stand-Alone Dump Program

Diagram SADMP-1. AMDSAAID – Address Space Locater (Part 1 of 2)

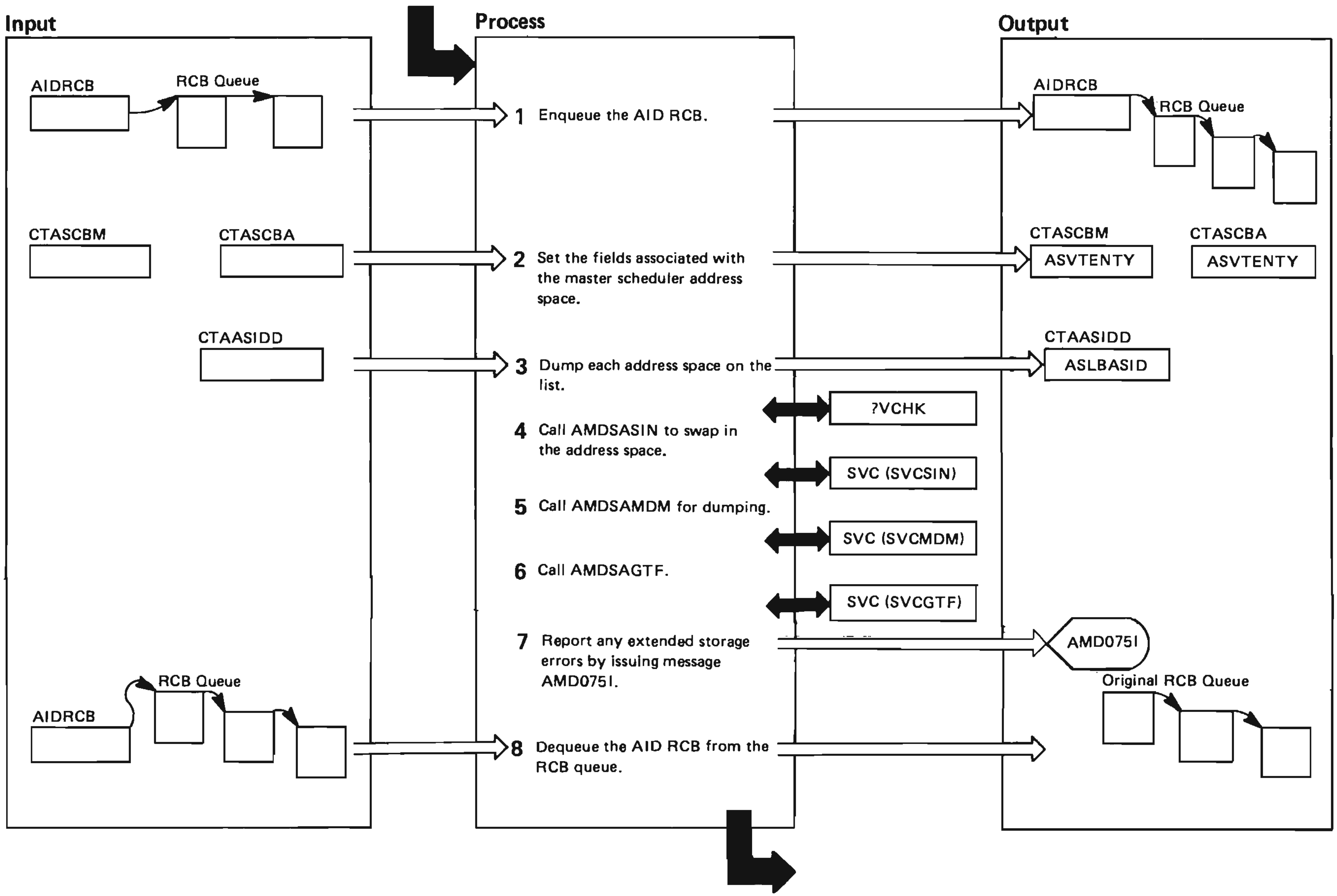


Diagram SADMP-1. AMDSAAID — Address Space Locator (Part 2 of 2)

Extended Description	Module	Label
1 AMDSAAID enqueues the AID RCB.		
2 AMDSAAID fills in fields CTASCBM and CTASCBA.		
3 AMDSAAID dumps each address space on the list of address spaces.		
4 AMDSAAID calls AMDSASIN to swap in the address space.		
5 AMDSAAID calls AMDSAMDM to dump the appropriate data areas.		
6 AMDSAAID calls AMDSAGTF to dump GTF virtual dump buffers.		
7 If SADMP detected any errors while dumping from extended storage, AMDSAAID writes message AMD075I to the operator console. This is done after all address spaces have been dumped and for each address space that had errors dumping from extended storage.		
8 AMDSAAID dequeues the AID RCB from the RCB queue.		

Diagram SADMP-2. AMDSAARD – Address Range Dump (Part 1 of 2)

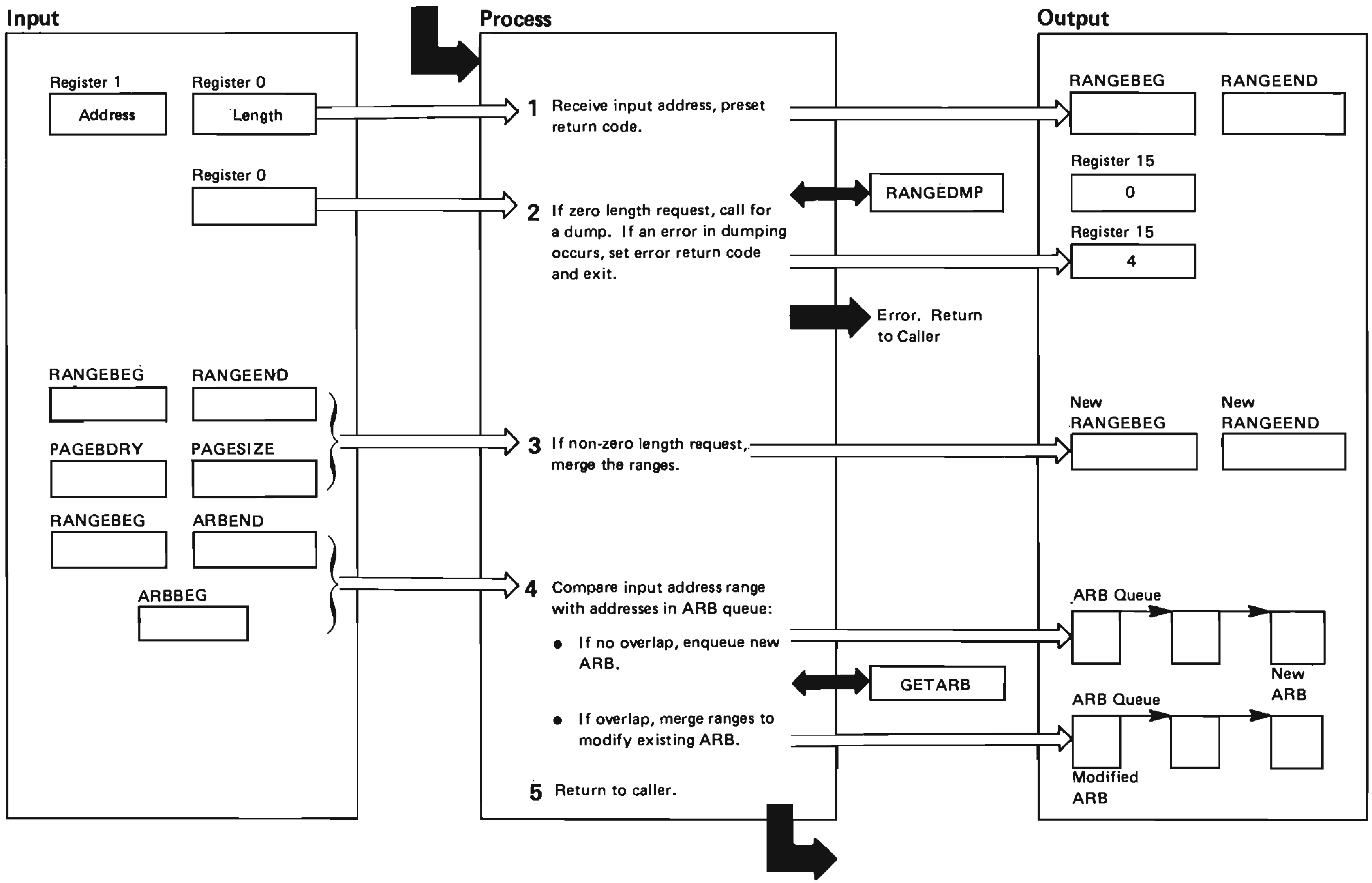


Diagram SADMP-2. AMDSAARD – Address Range Dump (Part 2 of 2)

Extended Description	Module	Label
AMDSAARD collects and merges the address ranges when overlaps occur, for zero length requests, AMDSAARD dumps each page within the range.		
1 AMDSAARD receives the input beginning address in register 1 and saves the beginning address in RANGEBEG. AMDSAARD calculates the ending address and saves the ending address in RANGEEND. AMDSAARD presets the return code to zero.		
2 AMDSAARD determines the function to be performed according to the value in register 0. If the value is zero, AMDSAARD calls RANGEDMP to dump every page listed in the ARB queue. If the dump fails, RANGEDMP sets an error return code of 4 and terminates.		RANGEDMP
3 If register 0 contains a nonzero value, AMDSAARD merges the input address range with the address ranges on the ARB queue. The ARB queue is in ascending order. AMDSAARD examines the ARBs, starting at the beginning of the ARB queue.		
4 AMDSAARD compares RANGEBEG to the ARB beginning and ending addresses. If the input address range does not overlap with any ARBs, AMDSAARD calls GETARB to enqueue a new ARB containing the input address range. If the input address range does overlap with an existing ARB, AMDSAARD merges the input address range into that ARB. If AMDSAAUD skips over the entire ARB queue because all ARBs end before the input address begins, AMDSAARD adds the new ARB to the end of the ARB queue.		GETARB
5 AMDSAARD returns to the caller.		

Diagram SADMP-3. AMDSAASM – Auxiliary Storage Management (Part 1 of 4)

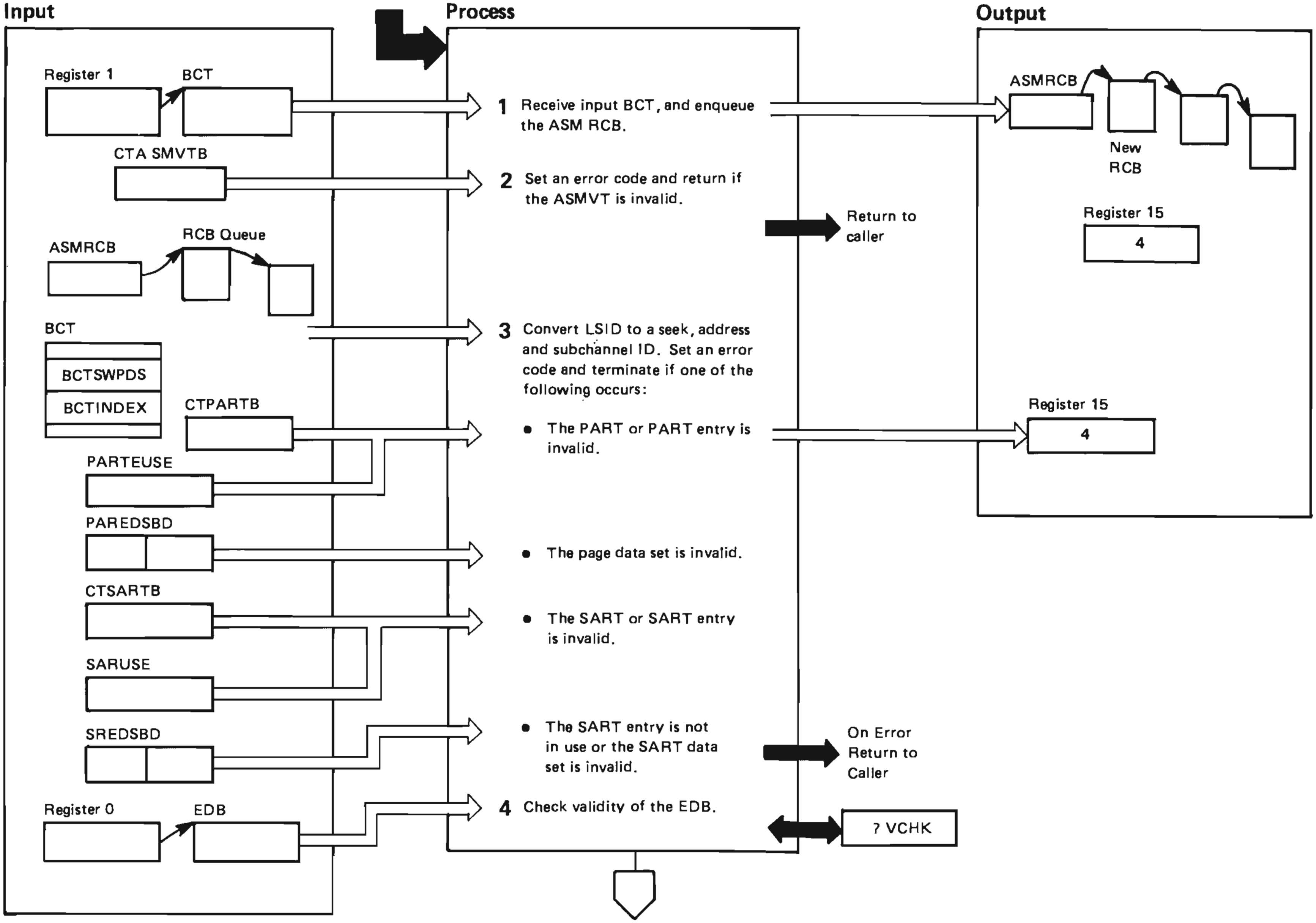


Diagram SADMP-3. AMDSAASM — Auxiliary Storage Management (Part 2 of 4)

Extended Description	Module	Label
AMDSAASM converts a logical slot identifier (LSID), which is the auxiliary storage ID for a page, into the direct access seek address of the auxiliary storage copy.		
1 AMDSAASM receives the input buffer control table (BCT), and enqueues an ASMRCB.		
2 If the CTASMVTB is on, an error code of 4 is set.		
3 AMDSAASM converts the LSID to a seek address and subchannel ID. For the following errors, AMDSAASM sets an error return code of 4, dequeues the ASMRCB, and terminates: ● The PART control block is invalid. ● The PART entry or page data set is invalid. ● The SART control block or SART entry is invalid. ● The SART entry is not in use. ● The SART data set is invalid.		
4 AMDSAASM calls ?VCHK to check the validity of the EDB. If it is invalid, ?VCHK sets a return code of 4 and AMDSAASM terminates.		?VCHK

Diagram SADMP-3. AMDSAASM – Auxiliary Storage Management (Part 3 of 4)

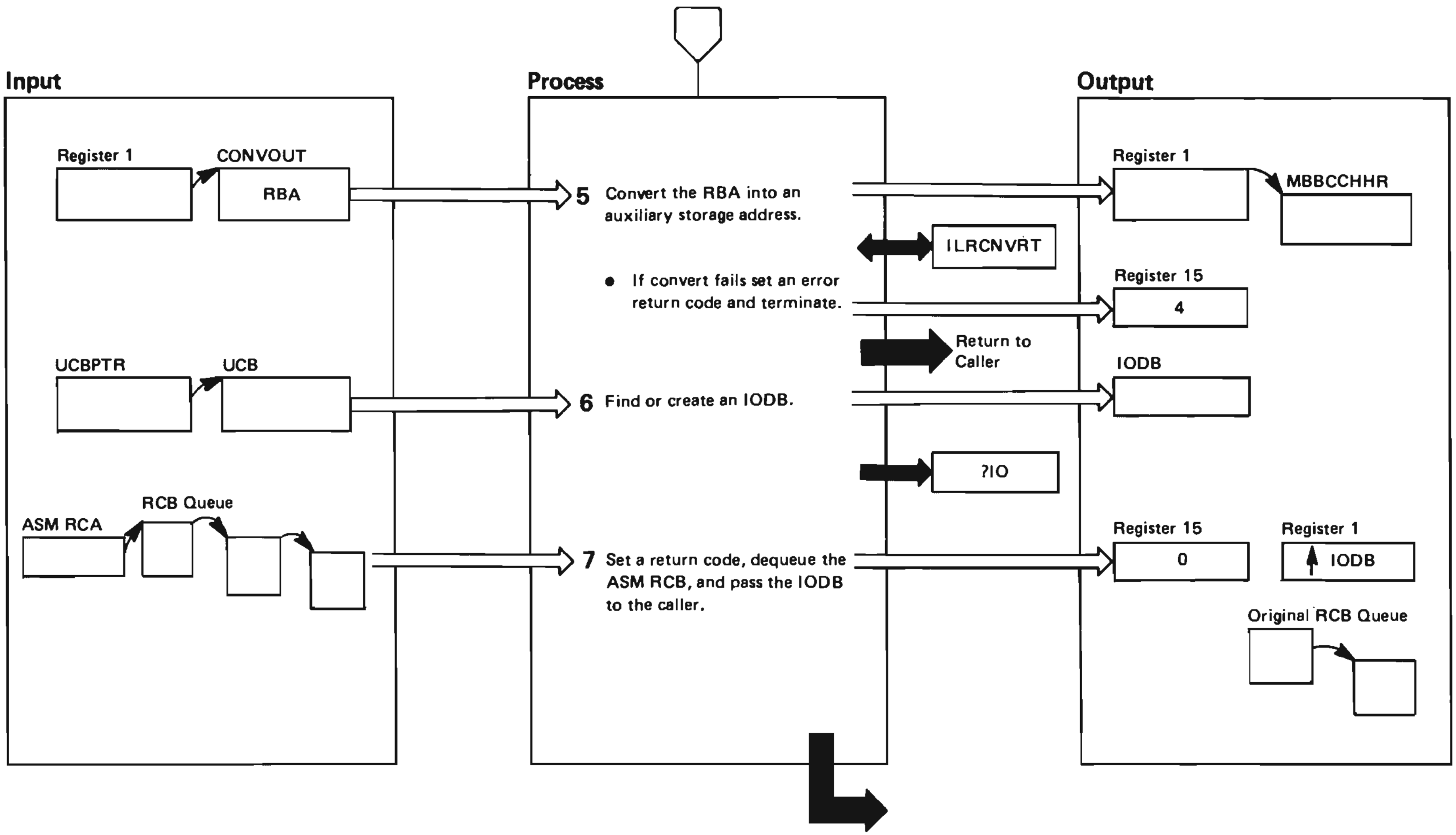
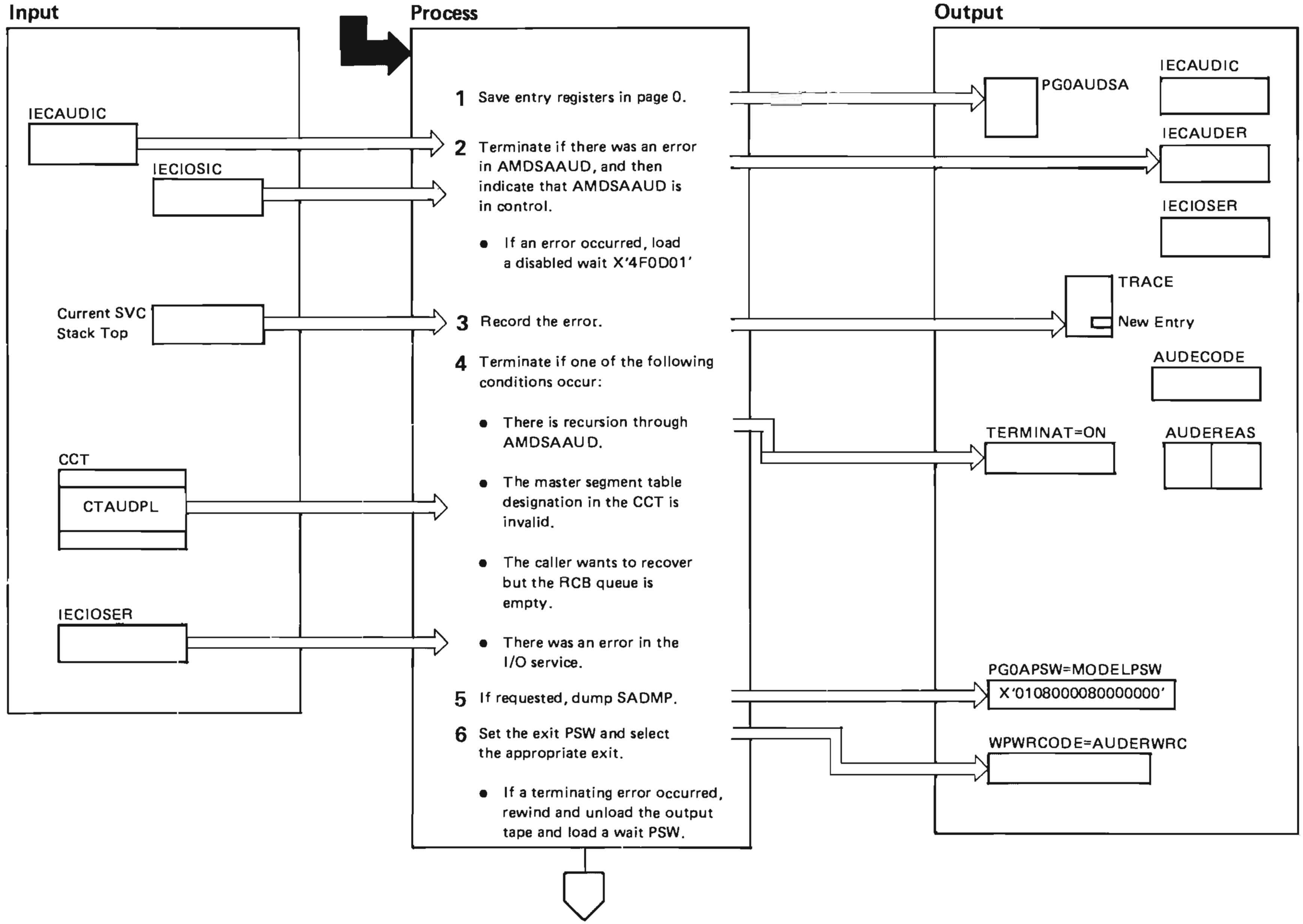


Diagram SADMP-3. AMDSAASM — Auxiliary Storage Management (Part 4 of 4)

Extended Description	Module	Label
5 AMDSAASM calls ILRCNVRT to convert the RBA into an auxiliary storage address (MBBCCHHR). ILRCNVRT sets a return code of 0 if the conversion is successful, or a return code of 4 if the conversion is not successful.		
6 AMDSAASM calls ?IO to find the IODB corresponding to the UCB. If the IODB does not exist, ?IO creates one.		?IO
7 AMDSAASM sets a return code of 0, dequeues the ASMRCB, and returns. Register 1 points to the IODB.		

Diagram SADMP-4. AMDSAAUD – Virtual Storage Dump Error Handling (Part 1 of 4)



1 Save entry registers in page 0.

2 Terminate if there was an error in AMDSAAUD, and then indicate that AMDSAAUD is in control.

- If an error occurred, load a disabled wait X'4F0D01'

3 Record the error.

4 Terminate if one of the following conditions occur:

- There is recursion through AMDSAAUD.
- The master segment table designation in the CCT is invalid.
- The caller wants to recover but the RCB queue is empty.
- There was an error in the I/O service.

5 If requested, dump SADMP.

6 Set the exit PSW and select the appropriate exit.

- If a terminating error occurred, rewind and unload the output tape and load a wait PSW.

Diagram SADMP-4. AMDSAAUD — Virtual Storage Dump Error Handling (Part 2 of 4)

Extended Description	Module	Label
1 AMDSAAUD saves the entry registers in page 0.		
2 AMDSAAUD examines the Internal Error Control (IEC) area to see if either AMDSAAUD or the SADMP I/O service was in control when the error occurred. AMDSAAUD sets IECAUDER = 1 if IECAUDIC = 1 and IECIOSER = 1 given IECIOSIC = 1. If the error was in AMDSAAUD (IECAUDER = 1 and IECIOSER = 0), AMDSAAUD immediately loads a disabled wait PSW.		IECAUDER
3 AMDSAAUD locates the appropriate SVC number and records the error.		
4 AMDSAAUD terminates if any of the following conditions exist: • There is an error in the I/O service (IECIOSER = 1). • The caller wants to recover but the RCB queue is empty (CTAUDRV = 1 and CTRCB = 0).		
5 AMDSAAUD dumps SADMP if a dump is requested or if a terminal error exists (CTAUDDMP = 1 or TERMINAT = 1).		

AMDSAAUD sets the exit PSW equal to X'0108000080000000' and selects an appropriate exit:

- If a terminating error occurred (CTAUDTRM = 1 or TERMINAT = 1), AMDSAAUD rewinds and unloads the output tape and loads a wait PSW.

Diagram SADMP-4. AMDSAAUD — Virtual Storage Dump Error Handling (Part 3 of 4)

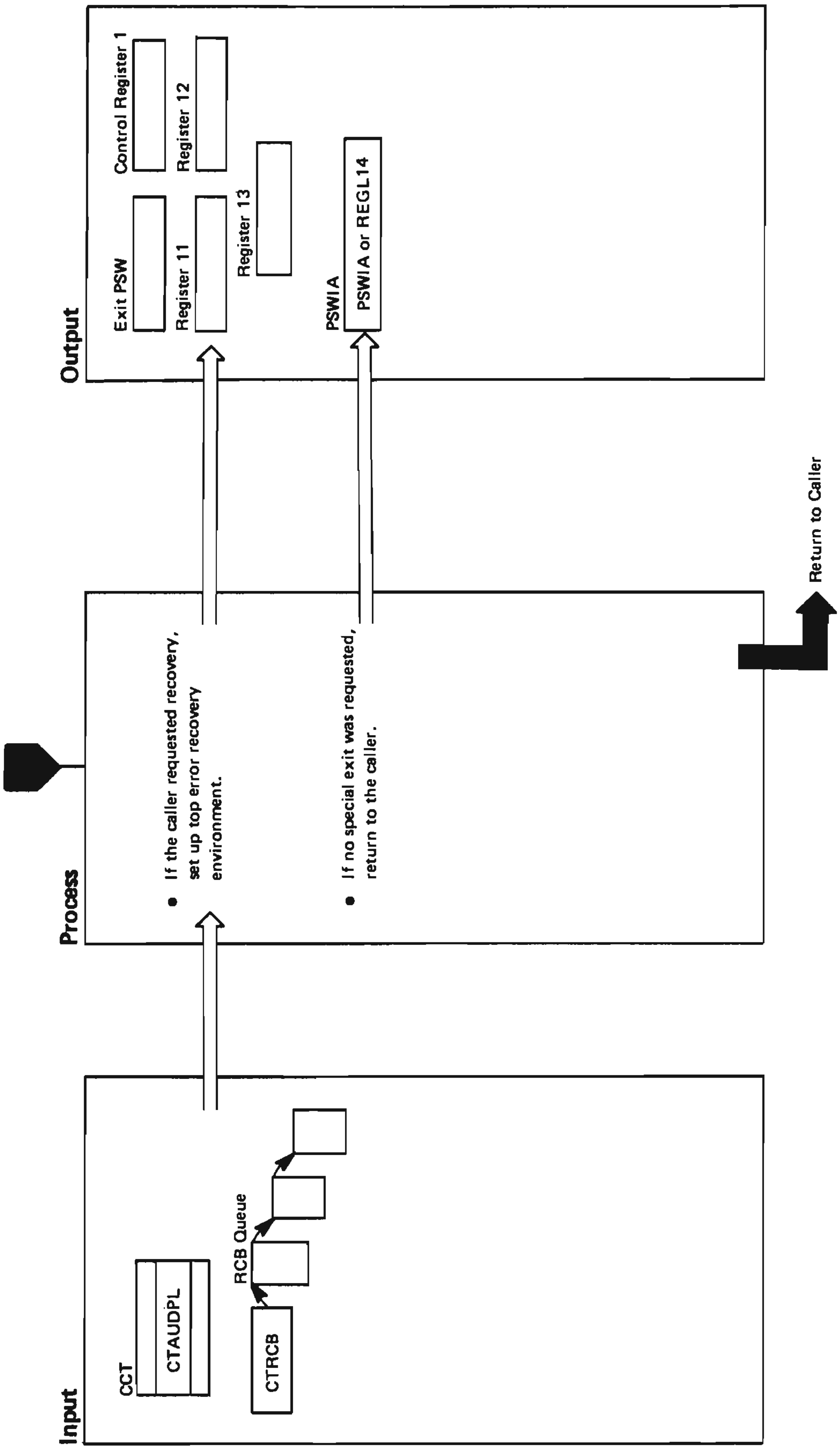


Diagram SADMP-4. AMDSAAUD — Virtual Storage Dump Error Handling (Part 4 of 4)

Extended Description	Module	Label
• If the caller requested recovery (CTAUDRCV = 1), AMDSAAUD sets up the top error recovery environment (RCB).		
• If no special exit was requested, AMDSAAUD returns to the caller.		

Diagram SADMP-5. AMDSABLD – Build Module (Part 1 of 10)

Input

SYSPRINT

Register 1

INPARMS

INPARMS

SYSPUNCH

IPLTEXT

IPLDEV

PGETEXT

For TYPE=HI

Process

1 Open SYSPRINT data set DCB.

2 Check the validity of the input parameters.

- If input parameters are invalid, set return code 8.

Error. Return to the Caller.

3 Open data set DCBs.

- If any data set DCB is not opened, return.

Return

Output

TYPE

HI or LOW

IPL

T or D

LOADPT

Register 15

8

MSG

AMD035I

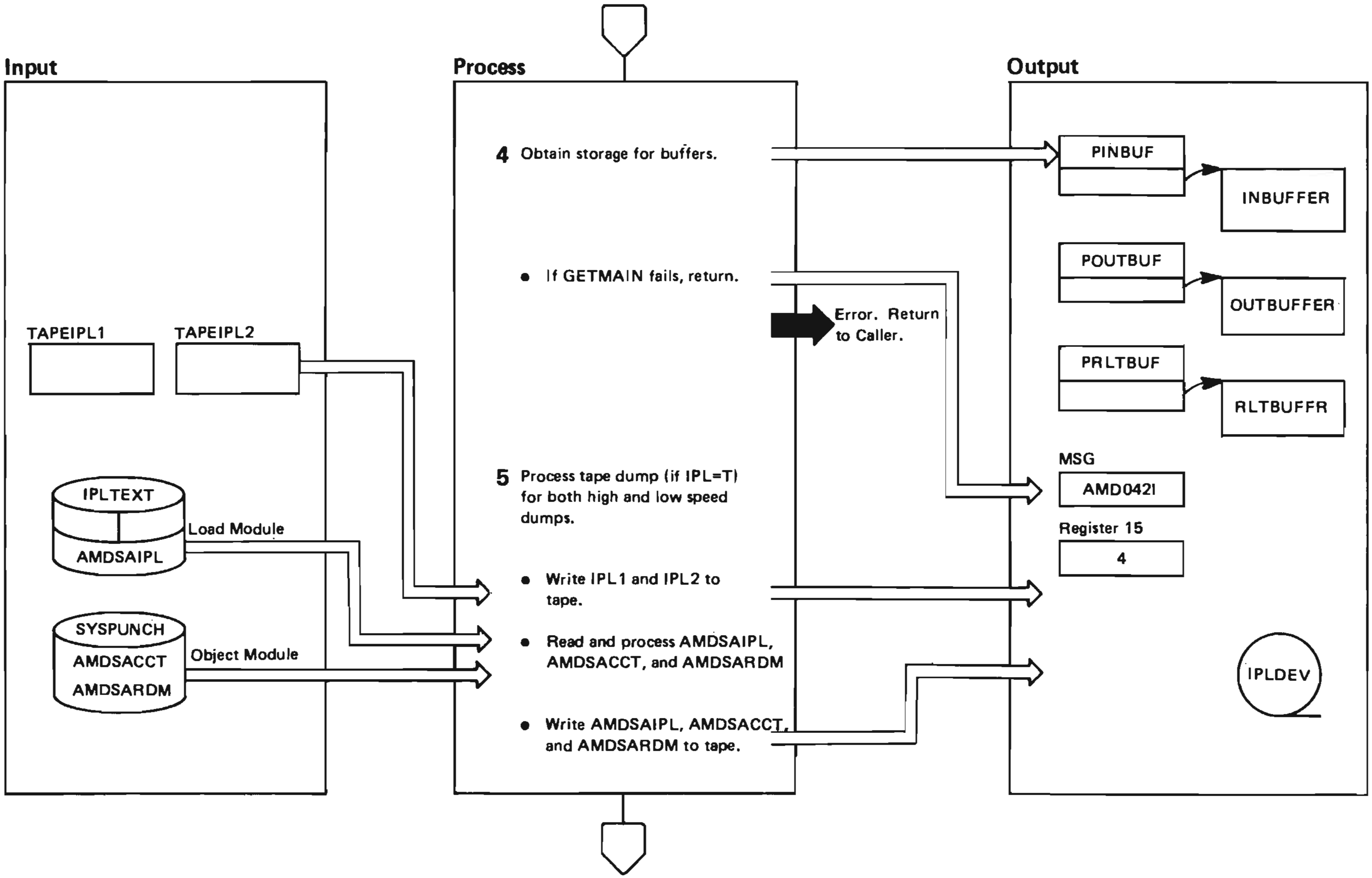
Register 15

4

Diagram SADMP-5. AMDSABLD — Build Module (Part 2 of 10)

Extended Description	Module	Label
AMDSABLD performs part of the initialization of the stand-alone dump program.		
1 AMDSABLD opens the SYSPRINT data set DCB so that error messages can be printed.		
2 AMDSABLD converts the input parameter, which is a character string, into three parameters; TYPE, IPL, and LOADPT. TYPE is the speed of the dump (HI or LOW). IPL specifies the unit type of the dump residence volume, either T for tape or D for direct access device. LOADPT is an eight-digit hexadecimal number representing the address in real storage at which AMDSARDM will be loaded. AMDXCB converts LOADPT from a hexadecimal character string into a binary number.		AMDXCB
AMDSABLD checks the validity of the three input parameters. If any of the input parameters are invalid, AMDSABLD sets the return code and calls AMDEXIT to terminate processing and return to the caller.		AMDEXIT
3 AMDSABLD opens the data set DCBs for SYSPUNCH, IPLTEXT, IPLDEV, and if TYPE=HI, for PGETEXT. If any open was unsuccessful, AMDSABLD calls AMDMESSG to print message AMD035I. Then AMDSABLD sets the return code and returns to the caller.		AMDMESSG

Diagram SADMP-5. AMDSABLD – Build Module (Part 3 of 10)



Extended Description	Module	Label
4 AMDSABLD issues GETMAINs to obtain storage for the input buffer, the output buffer, and the relocation table (RLT) buffer. INBUFFER is large enough to hold an input record of any length. OUTBUFFER is large enough to contain the storage that SADMP uses at run time. The RLT buffer is the area used for building the RLT. If the GETMAIN fails for any of the buffers, AMDSABLD calls AMDMESSG to print message AMD042I. AMDSABLD sets the return code and returns to the caller.		AMDMESSG
5 For tape dump processing of both high and low speed dumps, AMDSABLD calls AMDWRITE to write IPL1 and IPL2 on to the output tape (IPLDEV). AMDSABLD calls AMDTEXT to read and process AMDSAIPL, AMDSACCT, and AMDSARDM. If any errors occur in processing these modules, AMDSABLD sets the return code and returns to the caller. If no error occurs, AMDSABLD calls AMDWRITE to write AMDSAIPL, AMDSACCT, and AMDSARDM to the output tape (IPLDEV).		AMDWRITE AMDTEXT AMDWRITE

Diagram SADMP-5. AMDSABLD – Build Module (Part 5 of 10)

Input

PGETEXT
AMDSAPGE
Load Module

Process

6 If TYPE=HI, process the high speed dump, build the RLT, and write AMDSAPGE and the RLT to tape.

- If an error occurs in processing either high or low speed dump (for IPL=T), return.

Error. Return to the Caller.

Output

{AMDSAPGE RLT}

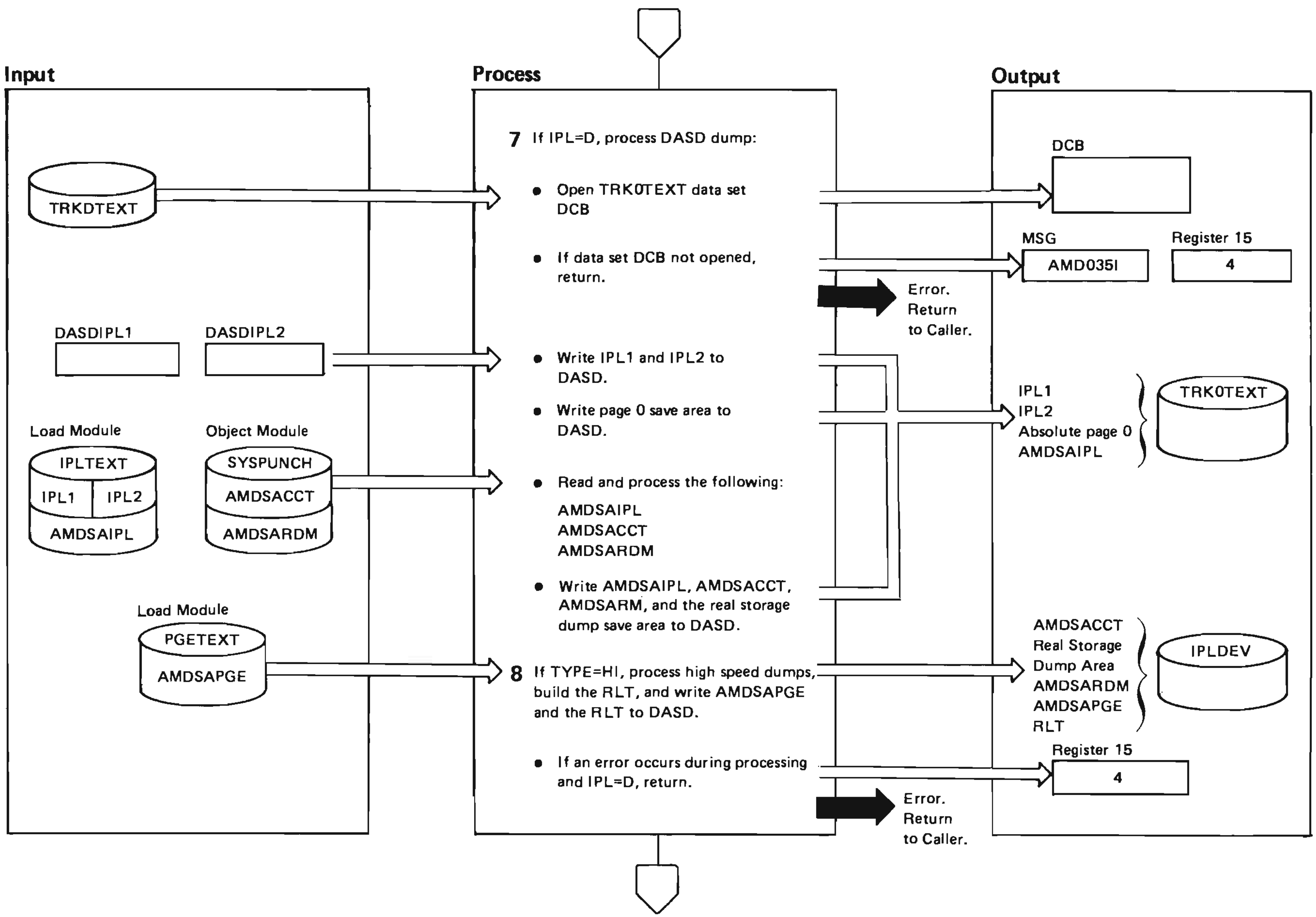
IPLDEV

Register 15

4

Extended Description	Module	Label
6 For tape dump processing of a high speed dump, AMDSABLD calls AMDTEXT to process AMDSAPGE and to build a relocation table. If any errors occur during processing, AMDSABLD sets the return code and returns to the caller. If no error occurs, AMDSABLD calls AMDWRITE to write AMDSAPGE and the RLT onto the output tape (IPLDEV).		AMDTEXT AMDWRITE

Diagram SADMP-5. AMDSABLD – Build Module (Part 7 of 10)



Extended Description	Module	Label
7 For DASD dump processing of both high and low speed dumps, AMDSABLD opens the TRK0TEXT data set DCB for temporary storage of the boot strap sequence. If the OPEN is unsuccessful, AMDSABLD calls AMDMESSG to print message AMD0351. AMDSABLD sets the error return code and returns to the caller. If the OPEN is successful, AMDSABLD calls AMDWRITE to write IPL1 and IPL2 to TRK0TEXT. AMDSABLD creates and empty record to be used by the IPL2 channel program to save absolute page zero. AMDSABLD calls AMDMOVE to clear the output buffer, and calls AMDWRITE to write absolute page zero to TRK0TEXT.		AMDMESSG AMDWRITE AMDMOVE
AMDSABLD calls AMDTEXT to read and process AMDSAIPL, AMDSACCT, and AMDSARDM. If any errors occur in the processing of these modules, AMDSABLD sets the error return code and returns to the caller. If no errors occur in processing, AMDSABLD calls AMDWRITE, AMDWRITE writes AMDSAIPL to TRK0TEXT, and writes AMDSACCT, AMDSARDM, and the real storage dump area to DASD IPLDEV.		AMDTEXT
8 For DASD dump processing of a high speed dump, AMDSABLD builds an RLT and calls AMDTEXT to process AMDSAPGE. If an error occurs in processing, AMDSABLD sets the error return code and returns to the caller. If no error occurs in processing, AMDSABLD calls AMDWRITE to write AMDSAPGE and the RLT to DASD IPLDEV.		AMDTEXT AMDWRITE

Diagram SADMP-5. AMDSABLD — Build Module (Part 9 of 10)

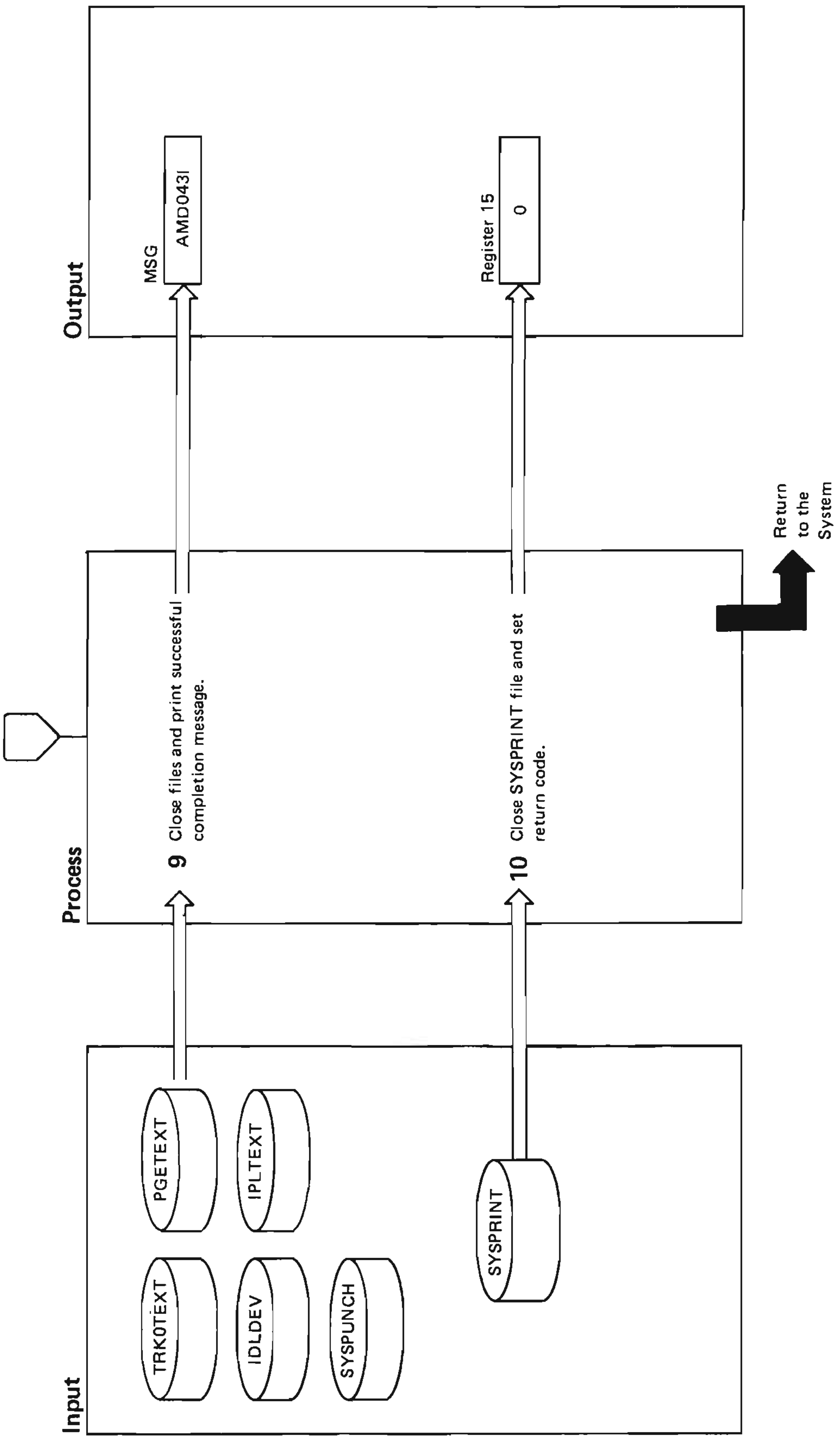


Diagram SADMP-5. AMDSABLD — Build Module (Part 10 of 10)

Extended Description	Module	Label
9 If the following files are open, AMDSABLD closes the files: TRK0TEXT, PGETEXT, IPLDEV, IPLTEXT, and SYSPUNCH. AMDSABLD prints message AMD0431 indicating successful completion of dump processing.		
10 If SYSPRINT is open, AMDSABLD closes the SYSPRINT file. AMDSABLD sets the return code and returns to the system.		

Diagram SADMP-6. AMDSABUF – BCT/Buffer Acquisition (Part 1 of 2)

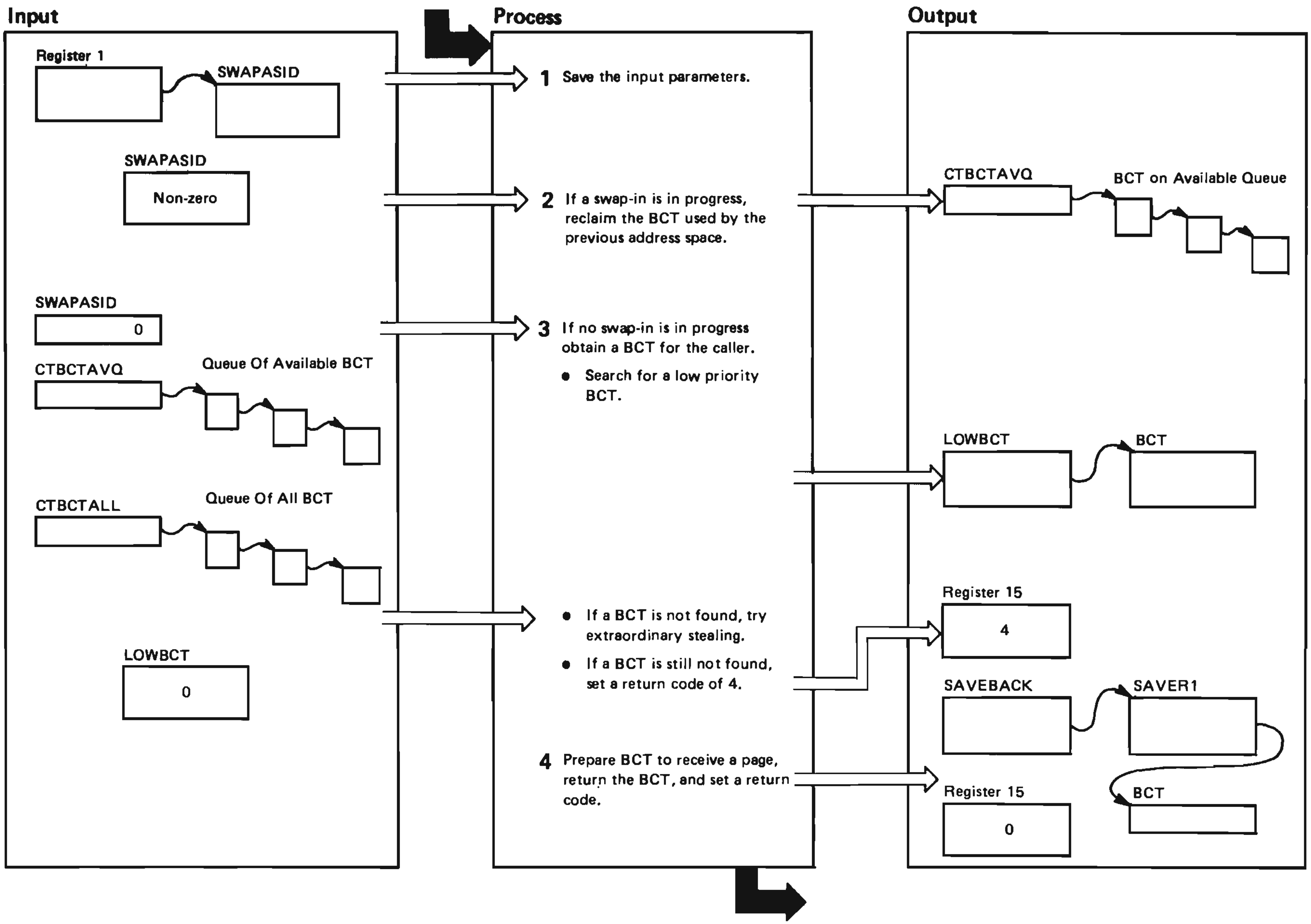


Diagram SADMP-6. AMDSABUF — BCT/Buffer Acquisition (Part 2 of 2)

Extended Description	Module	Label
1 AMDSABUF saves the input parameter in SWAPASID.		
2 If SWAPASID is nonzero, AMDSABUF assumes a swap-in is taking place and reclaims the BCT used by the previous address space unless: ● The BCT is already available. ● The BCT is already involved in paging. ● The BCT is for the swap-in address space.		
3 When no swap is occurring (SWAPASID = 0), AMDSABUF gets a BCT for the caller and searches the BCT for the lowest priority BCT to steal. If AMDSABUF cannot find a low priority BCT to steal, AMDSABUF waits for output I/O to complete, assuming that AMDSAIOI will make some BCT available or stealable upon completion of output I/O processing. If AMDSAIOI does not provide an available or stealable BCT, AMDSABUF sets an error return code of 4. AMDSABUF takes the first available BCT, dequeues it, and indicates that the BCT is involved in a paging operation. AMDSABUF returns the BCT address as a parameter and sets a return code of 0.		

Diagram SADMP-7. AMDSACON – Virtual Storage Dump Console Service (Part 1 of 2)

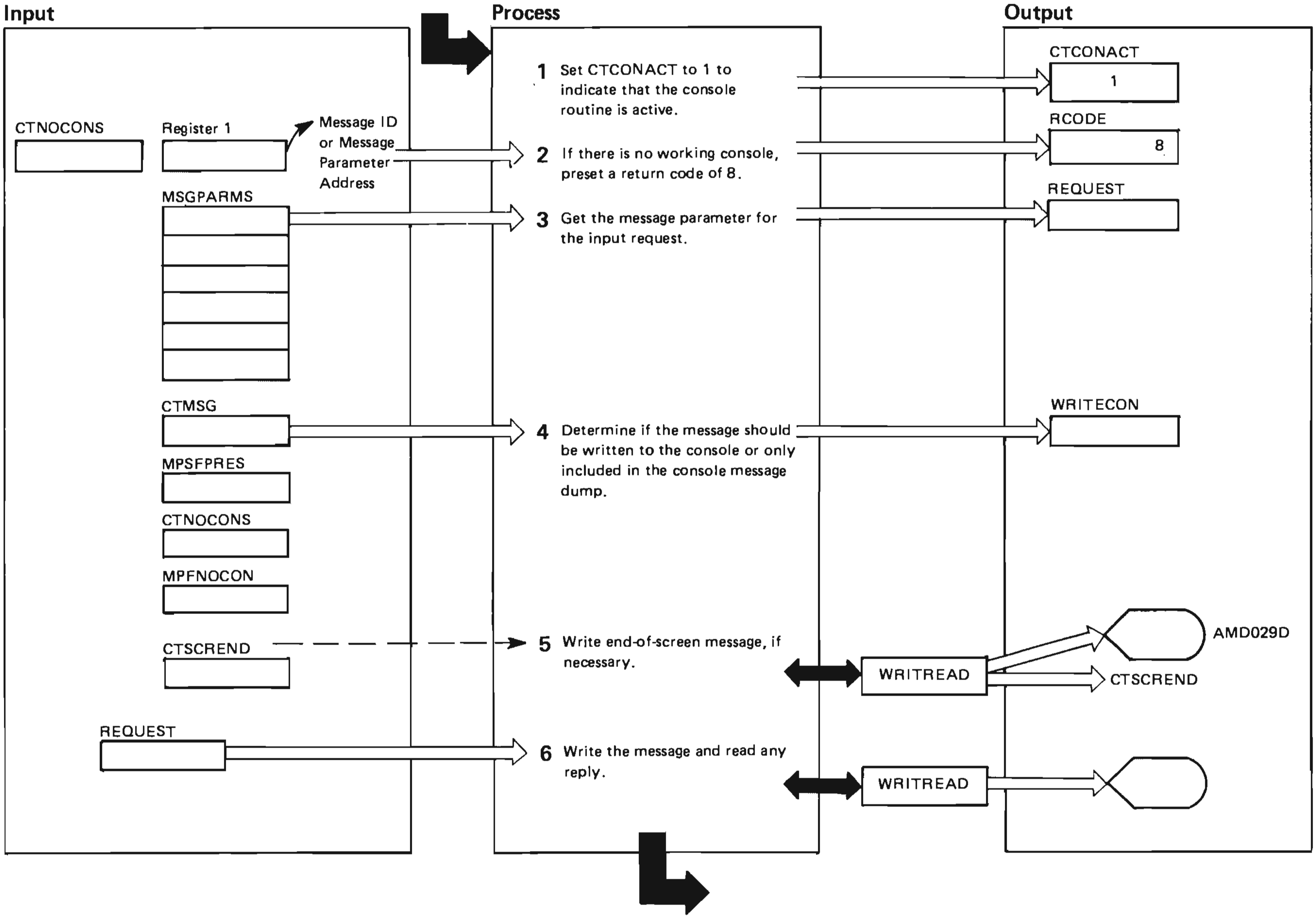


Diagram SADMP-7. AMDSACON — Virtual Storage Dump Console Service (Part 2 of 2)

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

1 AMDSACON sets CTCONACT = 1.

2 If CTNOCONS is on, AMDSACON sets the global return code RCODE to 8.

3 Message parameter control blocks (mapped by MSGPARM) describe SADMP messages. If register 1 at entry is greater than 255, it points to a message parameter. Otherwise, register 1 contains a SADMP message ID, which AMDSACON uses to index into a table (MSGPARMS) of message parameters. AMDSACON copies the message parameter into REQUEST.

4 Set WRITECON to 0 if:

- the message should be suppressed (CTMSG = 1 and MPFSPRES = 1)
- there is no console (CTNOCONS = 1)
- this message is only for the console message dump (MPFNOCON = 1)

Otherwise, WRITECON is set to 1.

5 AMDSACON checks if the message overflows the screen. If CTSCREND is 0, AMDSACON writes message AMD029D to ask the operator whether to pause and issue AMD029D again the next time the screen is full. If the reply is 'N', AMDSACON sets CTSCREND on. Subroutine WRITREAD erases the screen before it writes messages.

6 AMDSACON passes REQUEST to WRITREAD to write the message to the console and receive any reply.

Diagram SADMP-8. WRITREAD – Subroutine of AMDSACON (Part 1 of 2)

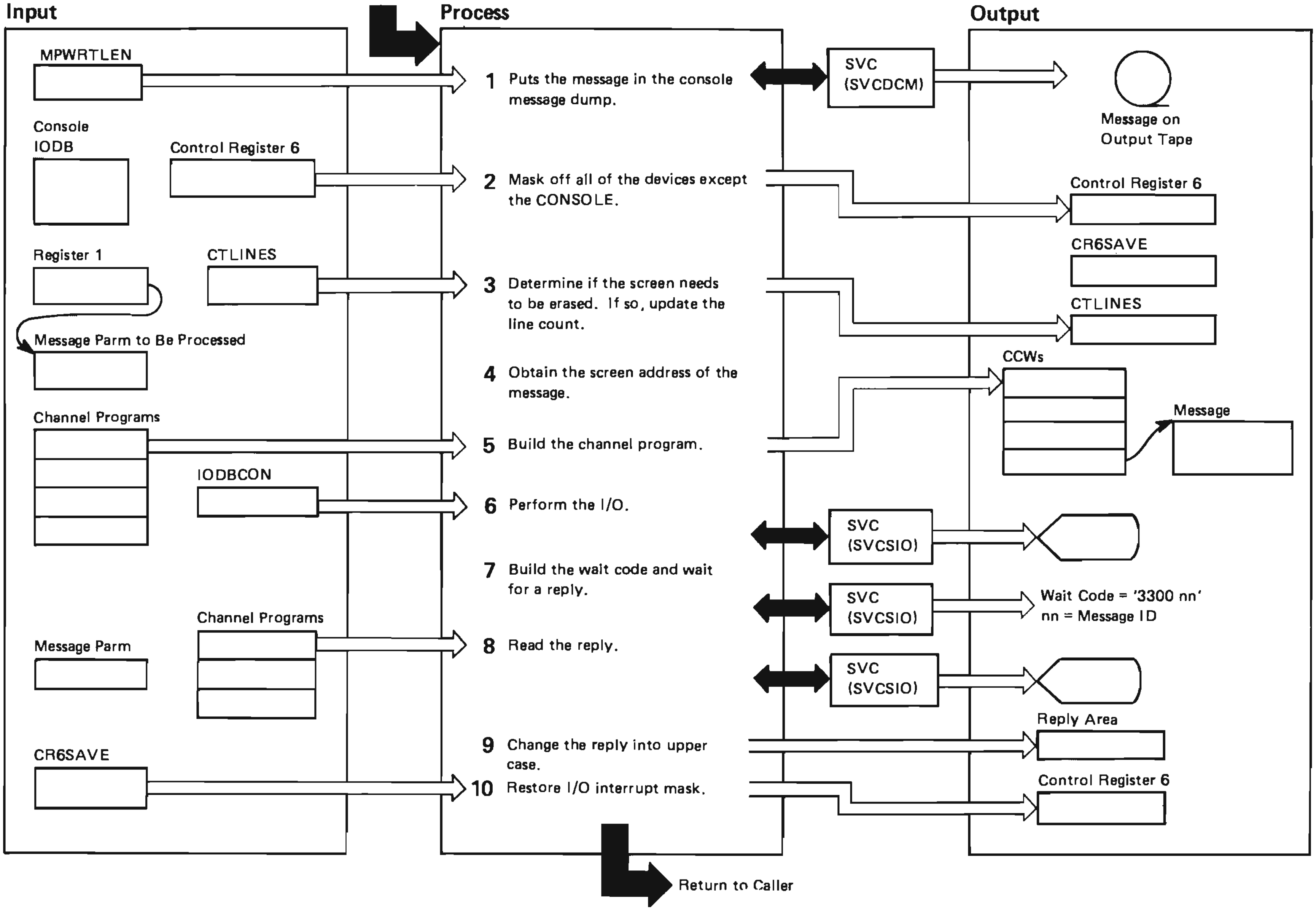


Diagram SADMP-8. WRITREAD — Subroutine of AMDSACON (Part 2 of 2)

Extended Description	Module	Label
1 If there is message text (MPWRTLEN is greater than 0), WRITREAD calls AMDSADCM to write the message to the output tape.		
2 WRITREAD changes the interrupt subclass mask in control register 6 to permit only console interrupts. This serializes all SADMP I/O until the message is complete. WRITREAD saves the original value of control register 6 in CR6SAVE.		
3 WRITREAD increases the count of lines on the screen. If the screen would be filled by that number of lines, WRITREAD sets the CCW command to ERASE/WRITE and resets CTLINES to 1.		
4 WRITREAD computes the 12/14-bit screen address from CTLINES and the known screen width (SW3277).		
5 The channel program sends a WRITE or ERASE/WRITE command to the console, unlocks the screen if a reply is expected, and furnishes the screen address and the data to be written.		
6 The ?IO macro points the console IODB (XIOBCON) to the channel program and calls AMDSASIO.		
7 If a reply is expected, WRITREAD calls AMDSASIO to wait for an attention interrupt caused when the operator presses enter.		
8 WRITREAD builds a channel program to read the reply and calls AMDSASIO (Steps 5 and 6).		
9 WRITREAD converts the reply to upper case.		
10 WRITREAD restores the original value of control register 6 and returns.		

Diagram SADMP-9. AMDSADER – DASD Error Recovery (Part 1 of 2)

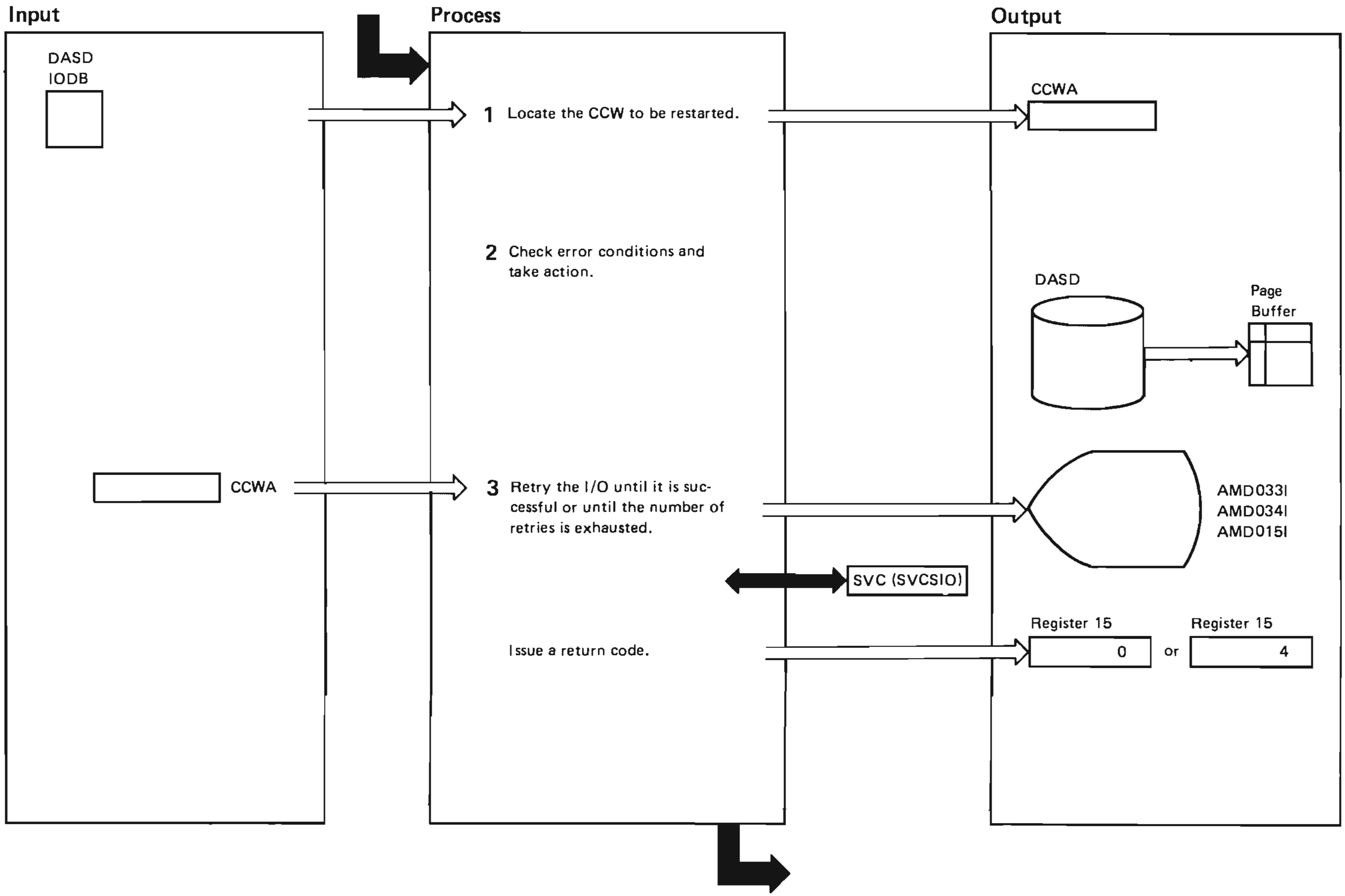


Diagram SADMP-9. AMDSADER — DASD Error Recovery (Part 2 of 2)

Extended Description	Module	Label

1 AMDSADER locates the CCW to be restarted. IODBCSW contains the last channel program address returned by an I/O interrupt. If this address is not 0, it is decreased by the length of a CCW to obtain the address of the failing CCW. If this address is 0, the address of the last CCW started (IODBCCWA) is used. IODBSCSW also contains the IRB status information.

If using a 3880-11 control unit (IODBPGST is on), the IODB is for a paging exposure. AMDSADER uses IODBBASA to locate the IODB for the base exposure. This IODB must be used in all error recovery.

2 The IRB status pointed to by IODBSCSW and the sense data pointed to by IODBSENS define the error and the action to be taken.

3 The CCWA contains the address of the CCW that is to be retried. AMDSADER calls AMDSASIO to retry the I/O, and performs Steps 1 through 3 until the I/O is successful, or the retry count is exhausted.

If the I/O succeeds, AMDSADER returns with a return code of 0. If the I/O does not succeed, AMDSADER writes I/O error message AMD033I to the console and returns with a return code of 4. If sense data is available, AMDSADER also writes message AMD034I. If there can be no more I/O with this device, AMDSADER marks the IODB as unusable (IODBUNAV = 1) and writes message AMD015I.

Diagram SADMP-10. AMDSADIP – Virtual Storage Dump Initialization (Part 1 of 4)

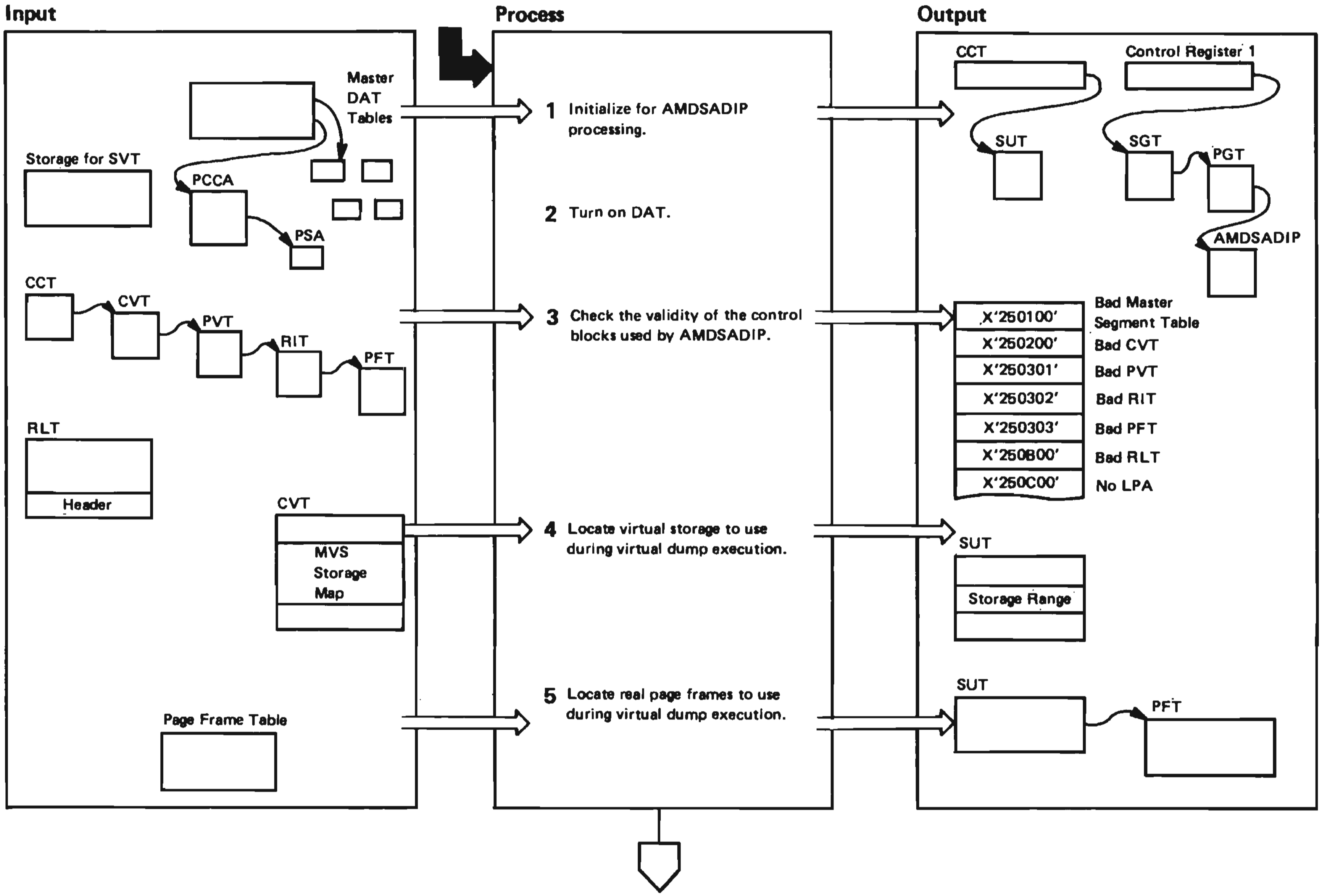


Diagram SADMP-10. AMDSADIP – Virtual Storage Dump Initialization (Part 2 of 4)

Extended Description	Module	Label

AMDSADIP is the first CSECT in load module AMDSAPGE, the virtual storage dump program. AMDSADIP loads the rest of AMDSAPGE into virtual and real storage, without overlaying any of the information AMDSAPGE needs to dump real storage.

1 AMDSADIP receives control with DAT off. AMDSADIP initializes the restart new PSW, the SVC new PSW, and the program check new PSW. AMDSADIP checks the validity of the segment table, and loads a wait state PSW if the segment table is invalid. AMDSADIP initializes the SUT and obtains the virtual address of itself.

2 AMDSADIP turns on DAT by passing control to an internal subroutine DATONOFF.

3 AMDSADIP checks the validity of the CVT, PVT, RIT, PFT, and RLT control blocks, and loads a wait state if any of the control blocks are invalid.

4 AMDSADIP locates virtual storage for SADMP to use during virtual dump processing. The storage must be contiguous, within the same segment, in COMMON, and free of information that SADMP might eventually use. AMDSADIP tries to obtain storage from non-directory PLPA or ELPA.

5 AMDSADIP locates real storage frames to back the virtual storage that it has just obtained. AMDSADIP takes real storage frames from the DAT-off nucleus to use for the segment table for swapped-in address spaces. AMDSADIP saves the addresses of usable frames in the SUT.

Diagram SADMP-10. AMDSADIP – Virtual Storage Dump Initialization (Part 3 of 4)

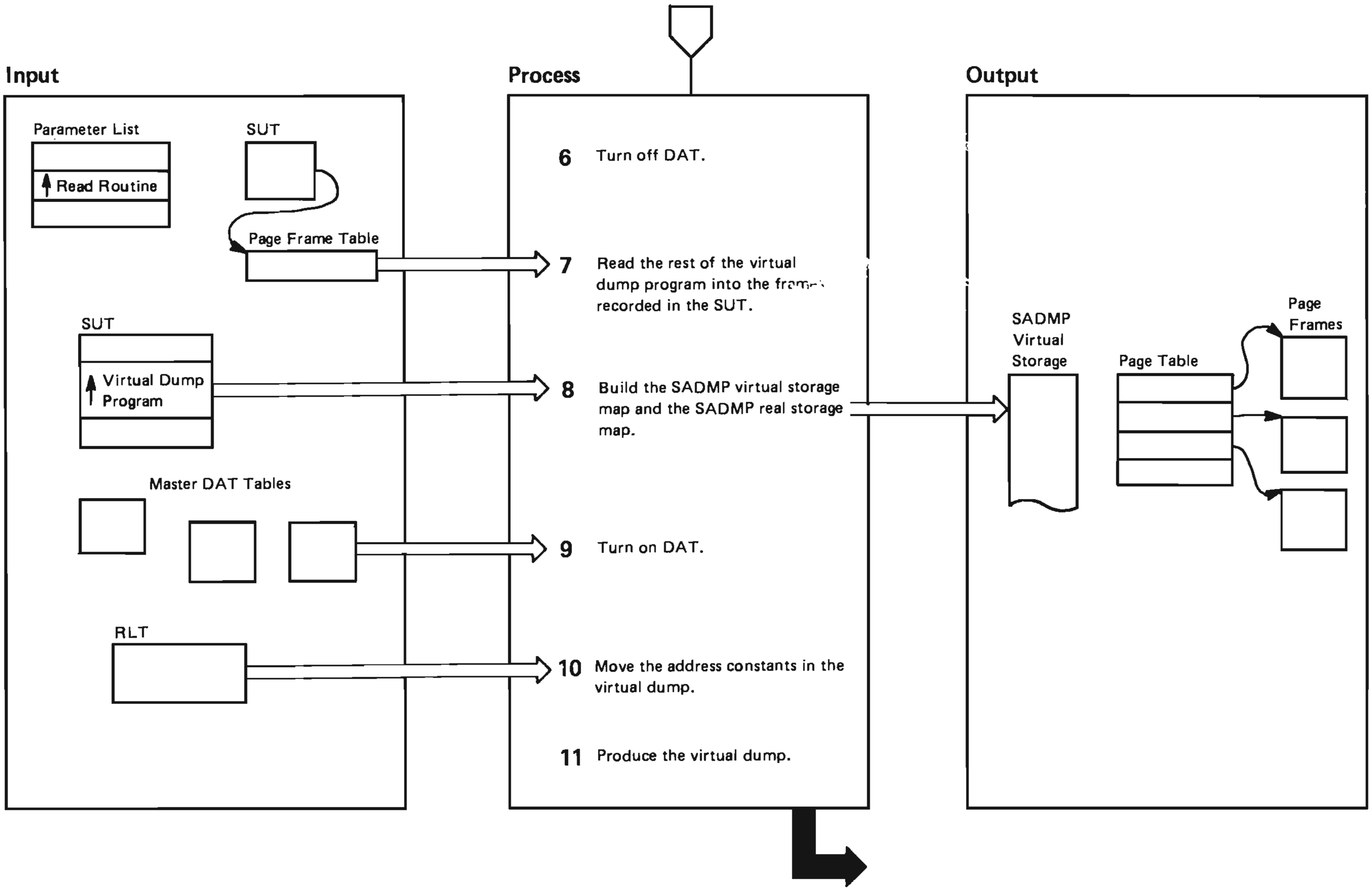


Diagram SADMP-10. AMDSADIP — Virtual Storage Dump Initialization (Part 4 of 4)

Extended Description	Module	Label
6 AMDSADIP turns off DAT by passing control to an internal subroutine DATONOFF.		
7 AMDSADIP calls a subroutine in AMDSAIPL to read the rest of the AMDSAPGE load module, 4K at a time, into the real page frames. If the read routine fails, the read routine loads a wait state PSW.		
8 AMDSADIP builds virtual storage and real storage maps for SADMP use.		
9 AMDSADIP turns on DAT by passing control to an internal subroutine DATONOFF.		
10 AMDSADIP uses the RLT and the SUT to relocate the address constants in AMDSAPGE.		
11 AMDSADIP calls AMDSAPGE to produce the virtual storage dump.		

Diagram SADMP-11. AMDSAEXI – External Interruption Handler (Part 1 of 2)

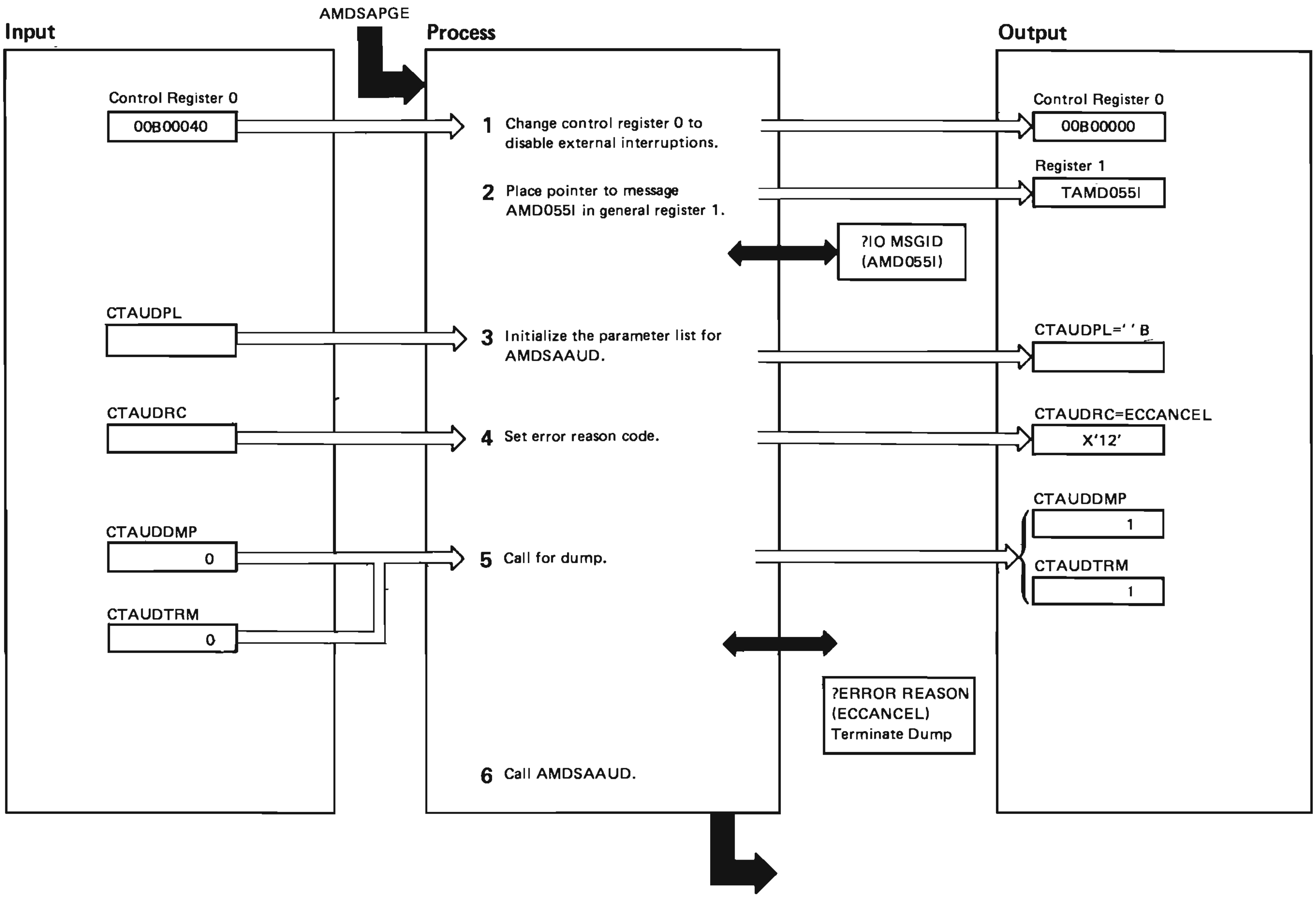


Diagram SADMP-11. AMDSAEXI – External Interruption Handler (Part 2 of 2)

Extended Description	Module	Label

AMDSAPGE calls AMDSAEXI (which handles external interruptions), writes message AMD055I, and loads a disabled wait PSW. Note: AMDSAMSF handles external interruptions itself by modifying the external new PSW.

1 AMDSAEXI changes control register 0, disabling external interruptions.

2 AMDSAEXI calls ?IO to issue message AMD055I. AMDSAEXI places the address of AMD055I in general register 1.

3 AMDSAEXI initializes the parameter list for AMDSAAUD. CTAUDPL is a field in the CCT.

4 AMDSAEXI sets the error reason code to X'12'.

5 AMDSAEXI initializes CTAUDDMP and CTAUDTRM (which are two fields in the CCT) and calls for a dump.

6 AMDSAEXI calls AMDSAAUD.

Diagram SADMP-12. AMDSAFRM – Storage Deallocation for Error Recovery (Part 1 of 2)

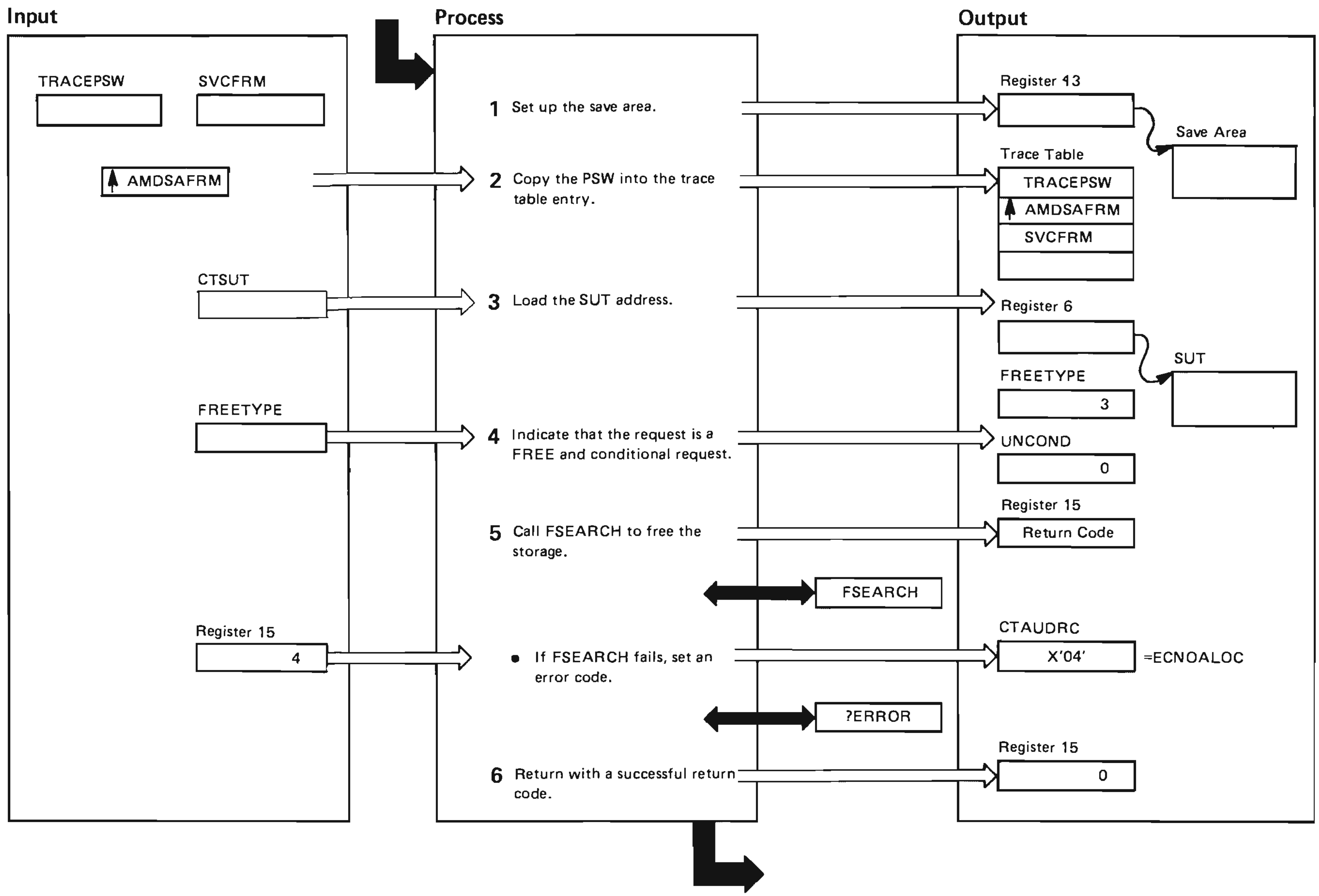


Diagram SADMP-12. AMDSAFRM — Storage Deallocation for Error Recovery (Part 2 of 2)

Extended Description	Module	Label
AMDSAFRM is a branch entry point for AMDSAAUD to recover storage resources not freed when ?ERROR causes entries to be deleted from the SVC stack.		
1 AMDSAFRM saves the passed save area address and sets up a local save area.		
2 AMDSAFRM calls ?TRACE to copy the PSW, the address of AMDSAFRM, and the SVC number for FRM into the trace table entry.		
3 AMDSAFRM loads the virtual address of the storage use table (SUT) into register 6 to expedite access to the SUT.		
4 AMDSAFRM sets FREETYPE=3 to indicate to FREE by SSINDEX, and sets off UNCOND to indicate a conditional free.		
5 AMDSAFRM calls FSEARCH to scan the DSCE queue for elements of allocated storage to be freed. If FSEARCH fails, AMDSAFRM issues a ?ERROR macro instruction to set an error code of X'04'.		
6 AMDSAFRM sets a successful return code of 0 and returns to the caller.		

Diagram SADMP-13. AMDSAGTF – GTF History Queue Dump (Part 1 of 4)

Input

GTFRCB — RCB Queue

MCHCUR — MCCE

GTFMCQEA — MCQE

MCHCUR Ø or GTFMCBEA Ø

or PCTPTR Ø

GTFASCBA — CTASCBD

GTFRCB — RCB Queue

GBCBPTR — GTEBCB

Process

1 Enqueue the GTFRCB

2 If GTF is not active, set an error return code.

?VCHK

- If the MCQE is invalid, set an error return code and issue a message.

?IO

3 Dequeue the GTFRCB.

4 If the ASCB is not for GTF, set an error return code.

5 Enqueue the GTFRCB.

6 Dump the GTF history queue.

Output

GTFRCB — New RCB

Register 15: 2

Register 15: 16

MSGID: AMD018I

Original RCB Queue

Register 15: 4

GTFRCB — New RCB

Diagram SADMP-13. AMDSAGTF – GTF History Queue Dump (Part 2 of 4)

Extended Description	Module	Label
1 AMDSAGTF enqueues a GTF RCB.		
2 AMDSAGTF calls ?VCHK to check the validity of the MCCE and MCQE to determine if GTF was active. If GTF was not active, AMDSAGTF sets an error return code of 12. If the MCQE is invalid, AMDSAGTF sets an error return code of 16 and calls ?IO to issue error message AMD018I.	AMDSAVCK	?VCHK ?IO
3 AMDSAGTF dequeues the GTF RCB.		
4 AMDSAGTF checks if the ASCB being dumped is for GTF. If GTF was not in this address space, AMDSAGTF sets an error return code of 4.		
5 AMDSAGTF enqueues a GTF RCB.		
6 AMDSAGTF dumps the GTF history queue.		

Diagram SADMP-13. AMDSAGTF – GTF History Queue Dump (Part 3 of 4)

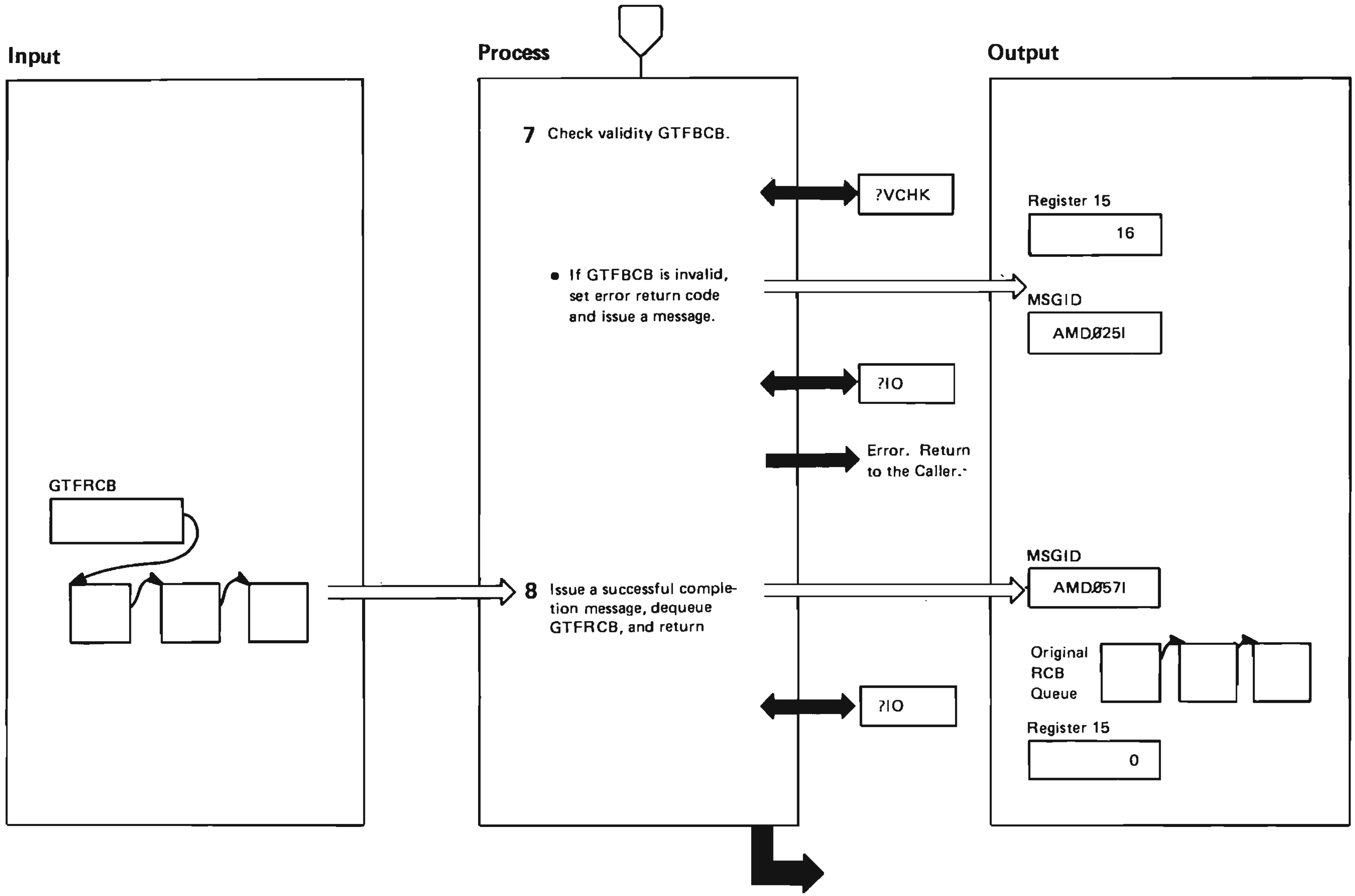


Diagram SADMP-13. AMDSAGTF – GTF History Queue Dump (Part 4 of 4)

Extended Description	Module	Label
7 AMDSAGTF calls ?VCHK to check the validity of the GTFBCB. If the GTFBCB is invalid, AMDSAGTF sets an error return code of 16, calls ?IO to issue error message AMD025I, and returns.		?VCHK ?IO
8 AMDSAGTF calls ?IO to issue completion message AMD057I, dequeues the RCB, and returns to the caller with a return code of 0.		?IO

Diagram SADMP-14. AMDSAGTM – Dynamic Storage Management (Part 1 of 6)

Input

Register 0 | Register 1

Register 0 | Register 15

CTSUT

GET: 1 | GETLEN

UNCOND: 1

UNCOND: 0

Process

1 If SSNUM=SVCGTM, save the input parameters and test for request type.

2 If SSNUM=SVCGET, save the input parameters and test for request type.

3 Load the SUT address.

4a For GET requests:

- If this is an unconditional request and too much storage is requested, set an error code and terminate.

?ERROR

Error, Return to Caller

- If this is a conditional request and too much storage is requested, set an error code and a return code.

?ERROR

Output

POOLLEN | SUBPOOLN | REQLEN2 | GET | UNCOND

SPNMODE | SUBPOOLN | REQLEN1 | GET | UNCOND

Register 6: SUTPTR → SUT

CTAUDRC: X'01' =ECBIGLEN

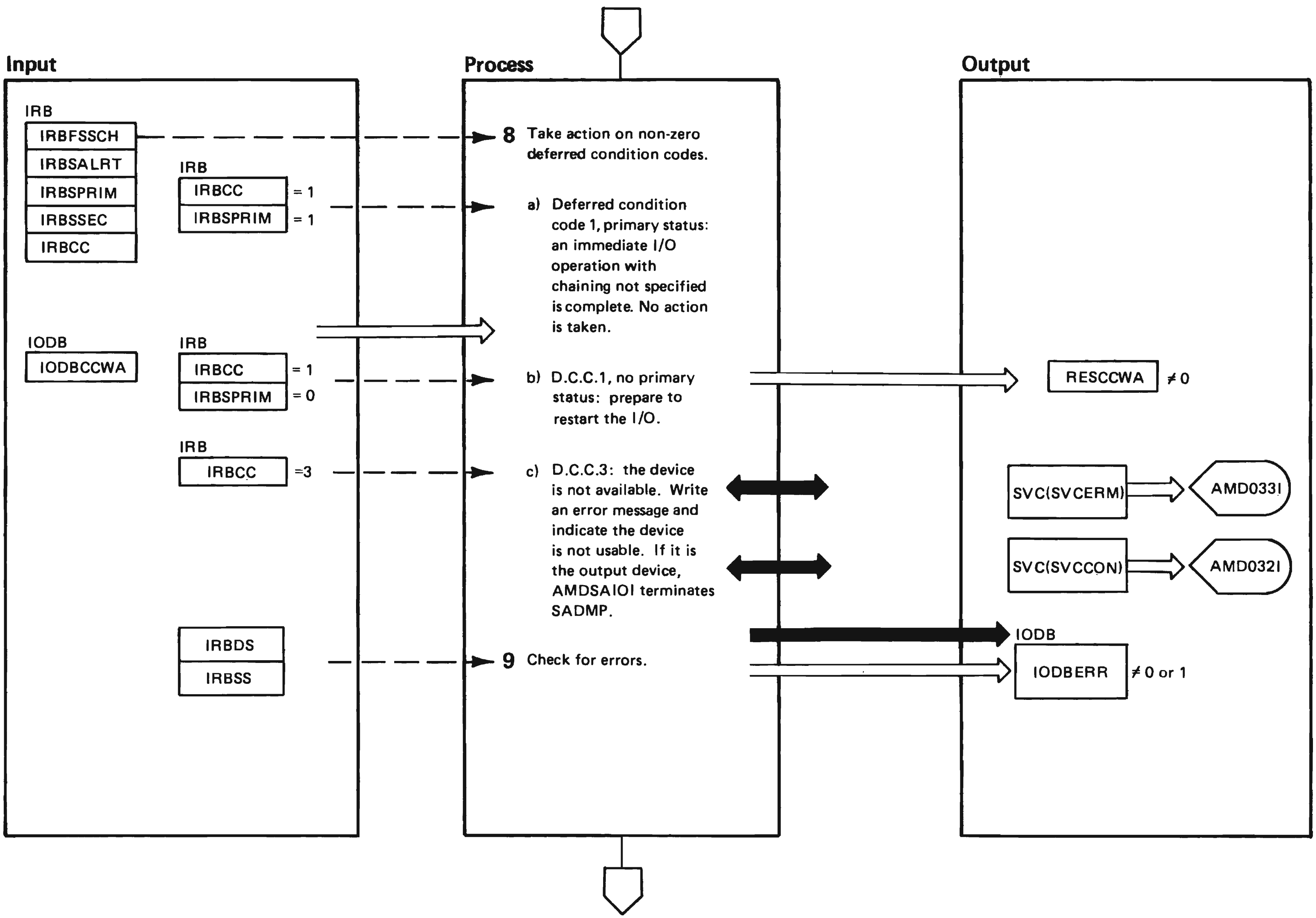
CTAUDRC: X'01' =ECBIGLEN

Register 15: 4

Diagram SADMP-14. AMDSAGTM — Dynamic Storage Management (Part 2 of 6)

Extended Description	Module	Label
1 For SSNUM=SVCGTM, AMDSAGTM saves the input parameters pointed to by register 0 and tests the value in register 1 to determine the request type. AMDSAGTM sets the GET field to 1 if register 1 contains a negative number. AMDSAGTM sets the UNCOND field to 1 because all R-form requests are unconditional.		
2 For SSNUM=SVCGET, AMDSAGTM saves the input parameters pointed to by registers 0 and 15. AMDSAGTM tests for the request type and sets the GET and UNCOND fields accordingly.		
3 AMDSAGTM loads the virtual address of the storage use table (SUT) into register 6 to expedite access to the SUT.		
4a For GET requests, AMDSAGTM performs GET processing. AMDSAGTM tests the requested length of storage. For unconditional requests for too much storage, AMDSAGTM calls ?ERROR to set an error code of X'01' and terminates. For conditional requests for too much storage, AMDSAGTM calls ?ERROR to set an error code of X'01' and a return code of 4, then continues processing.		

Diagram SADMP-15. AMDSAIOI – Virtual Storage Dump I/O Interruption Handler (Part 3 of 8)



Extended Description	Module	Label
8 a) The deferred condition code is valid if IRBFSSCH = 1, and if any of IRBSALRT, IRBSPRIM or IRBSSEC is 1. b) If RESCCWA = 0, I/O does not need to be restarted. If RESCCWA ≠ 0, AMDSAIOI will restart I/O later. c) AMDSAIOI calls AMDSAERM to issue AMD033I (I/O error message, indicates deferred condition code of 3). If the device is the output device (IODBOUT = 1), AMDSAIOI issues AMD032I to indicate a fatal error on the output tape, and then terminates SADMP with wait code X'4FFDF1'.		
9 AMDSAIOI examines the IRB subchannel status (IRBSS) and device status (IRBDS) for error conditions. If any are found, AMDSAIOI sets IODBERR = 1.		

Diagram SADMP-18. AMDSAIPL – Bootstrap Module (Part 5 of 6)

Input

IPLDEV
AMDSARDM

or

Process

- Read AMDSARDM.
  - If the read fails, set reason code

5 If TYPE=LO, dump real storage.

Call AMDSARDM

6 If TYPE=HI, read AMDSADIP.

- If the read fails, set reason code.

7 Restore real storage to its original contents.

8 Call AMDSADIP.

AMDSADIP

Output

AMDSARDM

WPREASON

X'0800' =WRCNORDM

WPREASON

X'0900' =WRCNODIP

Diagram SADMP-18. AMDSAIPL – Bootstrap Module (Part 6 of 6)

Extended Description	Module	Label
AMDSAIPL reads AMDSARDM into the storage area that was just copied into SYS1.PAGEDUMP. If the read for AMDSARDM fails, AMDSABLD sets a wait reason code of X'0800'.		
5 AMDSAIPL calls AMDSARDM to dump real storage. If the dump is a low speed dump, AMDSARDM does not return to AMDSAIPL.		
6,7 AMDSAIPL reads AMDSADIP if TYPE = HI. If the read for AMDSADIP fails, AMDSAIPL sets a wait reason code of X'0900'. If the read is successful, AMDSAIPL restores the contents of storage that was occupied by AMDSARDM.		SIORTN
8 AMDSAIPL calls AMDSADIP. AMDSADIP loads and initialized AMDSAPGE; AMDSAPGE dumps virtual storage.	AMDSADIP	

Diagram SADMP-19. AMDSAMDM – Virtual Storage Dump Memory Dump (Part 1 of 4)

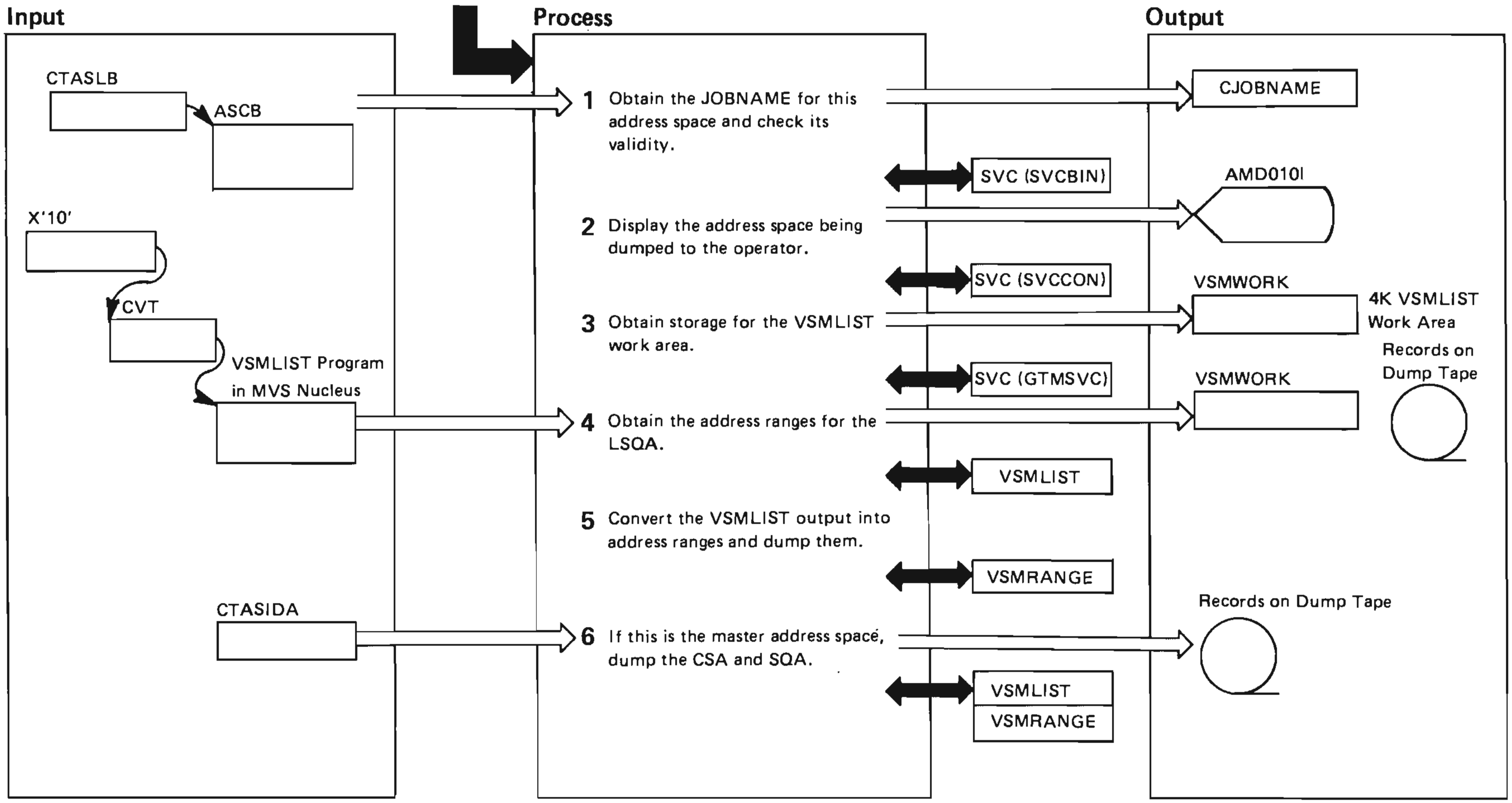


Diagram SADMP-19. AMDSAMDM – Virtual Storage Dump Memory Dump (Part 2 of 4)

Extended Description	Module	Label
1 AMDSAMDM obtains the jobname from ASCBJBNI or ASCBJBNS. If neither field contains a valid jobname, AMDSAMDM substitutes *UNKNOWN.		
2 AMDSAMDM calls AMDSABIN to convert the ASCB address and ASID to EBCDIC, then writes message AMD010I to the console.		
3 AMDSAMDM issues a GETMAIN for 4K of subpool 104. VSMLIST uses this as a work area.		
4 AMDSAMDM issues a VSMLIST macro call to put address ranges for the LSQA of the current address space into the VSMLIST work area.		
5 AMDSAMDM calls internal subroutine VSMRANGE. VSMRANGE processes the output from VSMLIST. VSMRANGE calls AMDSAARD (via an SVC macro) for each block of storage that VSMLIST identifies, and passes the length and address of each block to AMDSAARD. AMDSAARD accumulates these storage ranges and dumps the associated storage.		
6 If CTASID=1, AMDSAMDM issues VSMLIST macros for CSA and SQA, then calls VSMRANGE.		

Diagram SADMP-19. AMDSAMDM – Virtual Storage Dump Memory Dump (Part 3 of 4)

Input

CTDT

Dump Table

CTASIDD

CJOBNAME

ASXB

ASXBLTCB

TCB

Process

7 Search each entry in the dump table for the following conditions:

- The current ASID is in the ASID list.
- The current JOBNAME is in the JOBNAME list.
- The protect key of the memory's RCB is in the key list.

8 If any of the above conditions are found, do the following:

- Dump the address ranges in the address list.

SVC (SVCARD)

- Dump the subpools in the subpool list for every TCB in this memory.

TCBSCAN

Return to Caller

Output

ASID IB+

JOBNAME IB+

Key List

Address List

Subpool List

Tape

VSMWORK

VSMLIST

Tape

Diagram SADMP-19. AMDSAMDM – Virtual Storage Dump Memory Dump (Part 4 of 4)

Extended Description	Module	Label
7 The dump table (DTABLE, that the CTDT points to) contains lists of sublists that specify storage to be dumped (by subpool number or address range) in address spaces (by ASID, jobname, or TCB key). AMDSAMDM searches each sublist to determine if one of the following three conditions is met:		
• The address space currently being dumped is in the ASID list.		
• The jobname for the address space currently being dumped is in the jobname list.		
• The protect key of the memory's TCB for the address space currently being dumped is in the key list.		
If the address space meets the criteria that the dump table specifies, AMDSAMDM dumps the necessary storage. If address ranges were specified, AMDSAMDM calls AMDSAARD (via an SVC macro) to dump the address ranges. If subpools were specified, AMDSAMDM calls TCBSCAN to dump the subpools in the subpool list for each TCB in this address space.		

Diagram SADMP-20. AMDSAMSF – Console Loop Trace Buffer Dump (Part 1 of 6)

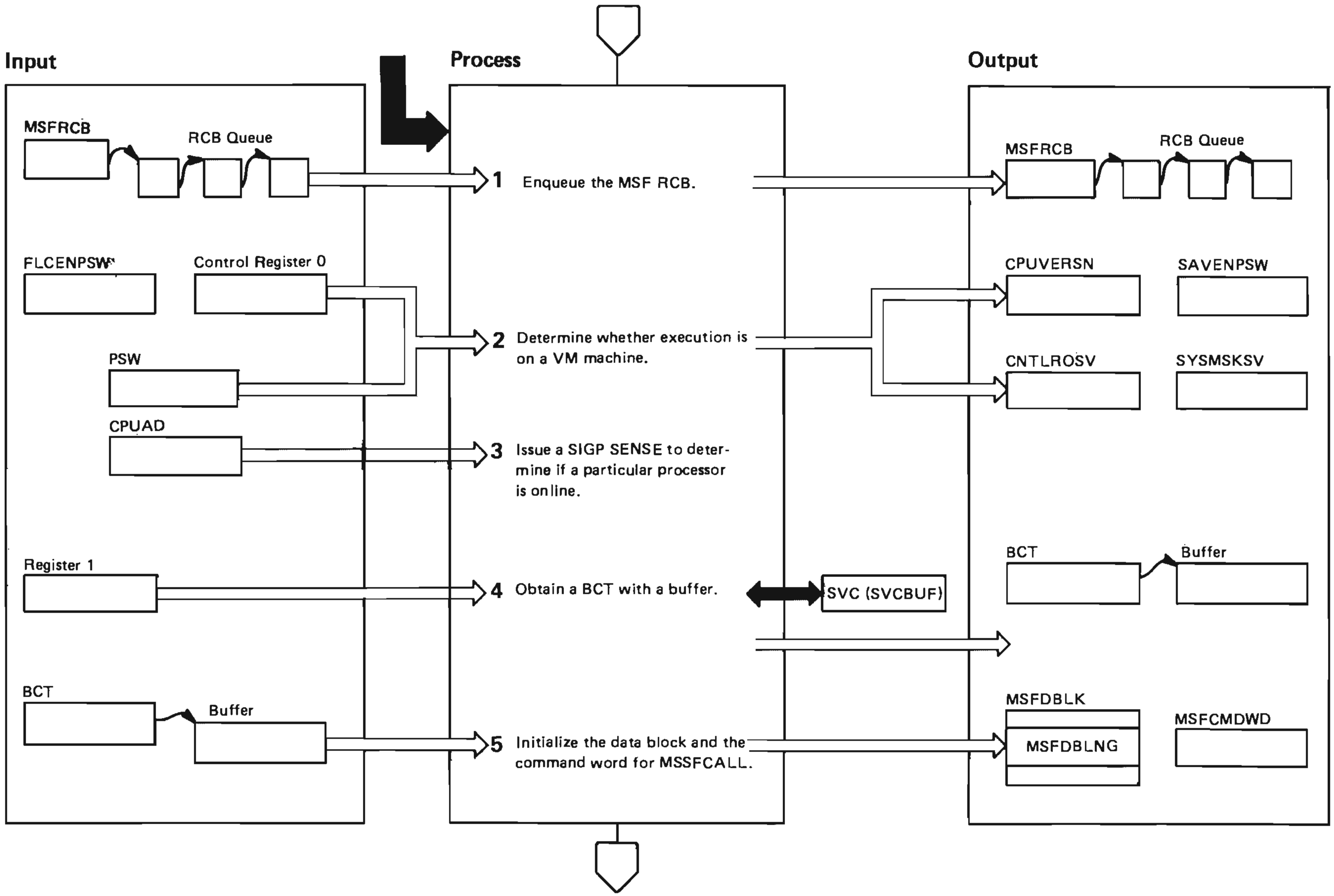
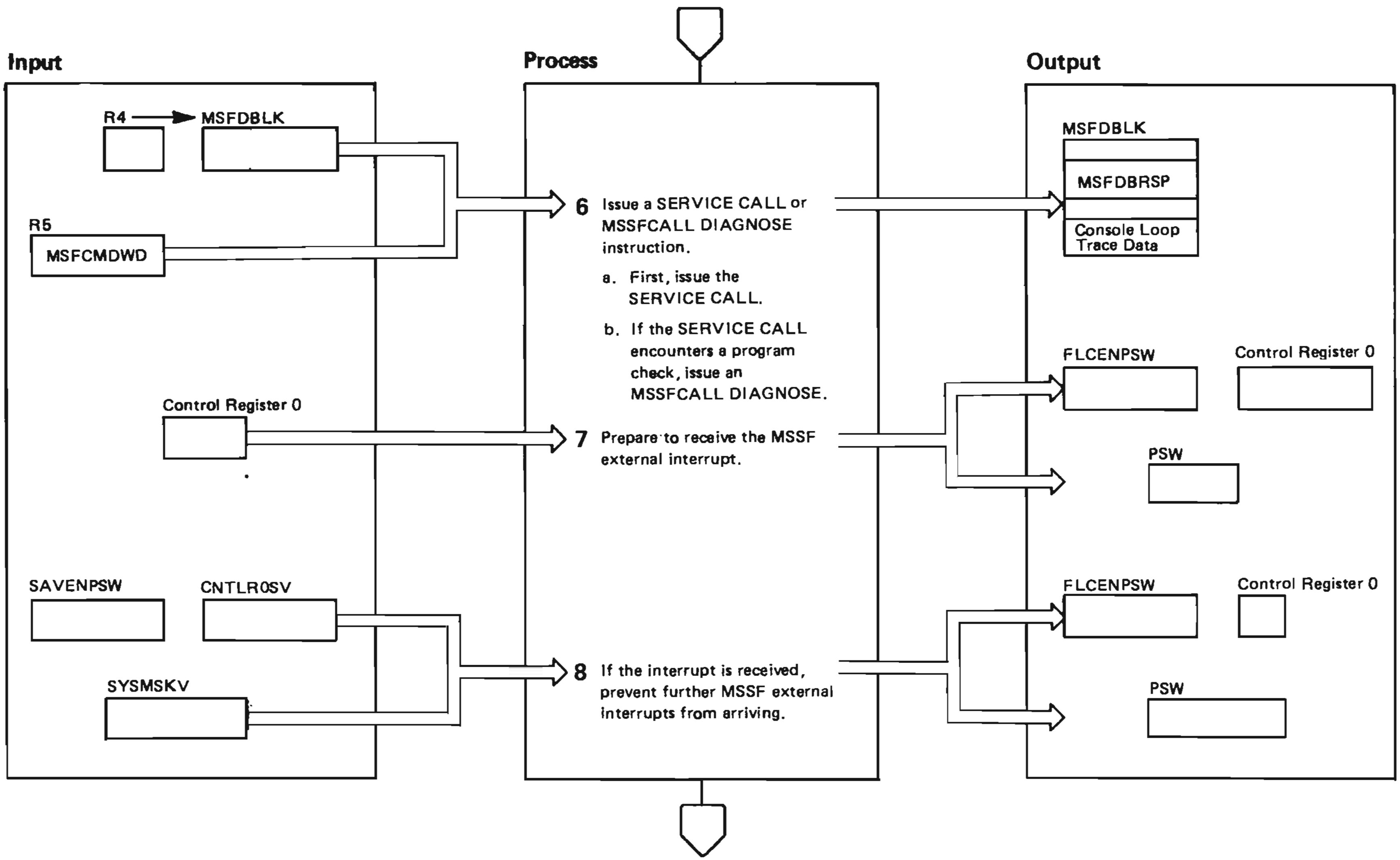


Diagram SADMP-20. AMDSAMSF – Console Loop Trace Buffer Dump (Part 2 of 6)

Extended Description	Module	Label
1 AMDSAMSF enqueues the MSF RCB onto the RCB queue.		
2 AMDSAMSF stores the processor identifier, to determine whether or not execution is on a VM machine. If execution is not on a VM machine. AMDSAMSF saves its environment. Otherwise, AMDSAMSF passes control to step 12.	AMDSAMSF	
3 AMDSAMSF issues SIGP instruction to a particular processor to determine if the processor is on line. If it is on line, AMDSAMSF continues processing. If the processor is not on line, AMDSAMSF does not perform steps 4 through 9.		
4 AMDSAMSF obtains a buffer control table (BCT) entry and its associated buffer by passing control to AMDSAMSF via an SVC instruction. If AMDSAMSF cannot obtain a buffer, it passes control to step 11.		
5 AMDSAMSF clears the BCT buffer for use as the MSSFCALL data block. It saves the length of the data block in the block. AMDSAMSF initializes the MSSFCALL command word so that a read console loop trace command is issued to the MSSF.		

Diagram SADMP-20. AMDSAMSF – Console Loop Trace Buffer Dump (Part 3 of 6)



	Extended Description	Module	Label
6	AMDSAMSF passes the command word and the address of the data block in registers that the SERVICE CALL and DIAGNOSE instructions use. AMDSAMSF issues the SERVICE CALL or MSSFCALL DIAGNOSE instruction to obtain the console loop trace buffer. It issues SERVICE CALL first; if SERVICE CALL encounters a program checks, it then issues MSSFCALL DIAGNOSE. If AMDSAMSF is successful, the console loop trace is stored in the data block passed to the MSSF.		
7	AMDSAMSF changes the new external interrupt PSW (FLCENPSW), which passes control to EIHANDLER in AMDSAMSF when an external interrupt occurs. AMDSAMSF modifies control register 0 to allow MSSF external interrupts and modifies the PSW to enable for external interrupts. It then goes into a spin loop to wait for the interrupt. If the TOD clock value is usable, AMDSAMSF waits a maximum of 60 seconds for the interrupt. If no interrupt occurs, AMDSAMSF stops attempting to obtain the console loop trace and passes control to step 11. If the TOD clock is not usable, AMDSAMSF is in an endless spin loop.		
8	The MSSF interrupt occurred. AMDSAMSF restores the original contents of the new external interrupt PSW and control register 0. It then disables the PSW to prevent further MSSF external interrupts.		EIHANDLER

Diagram SADMP-20. AMDSAMSF – Console Loop Trace Buffer Dump (Part 5 of 6)

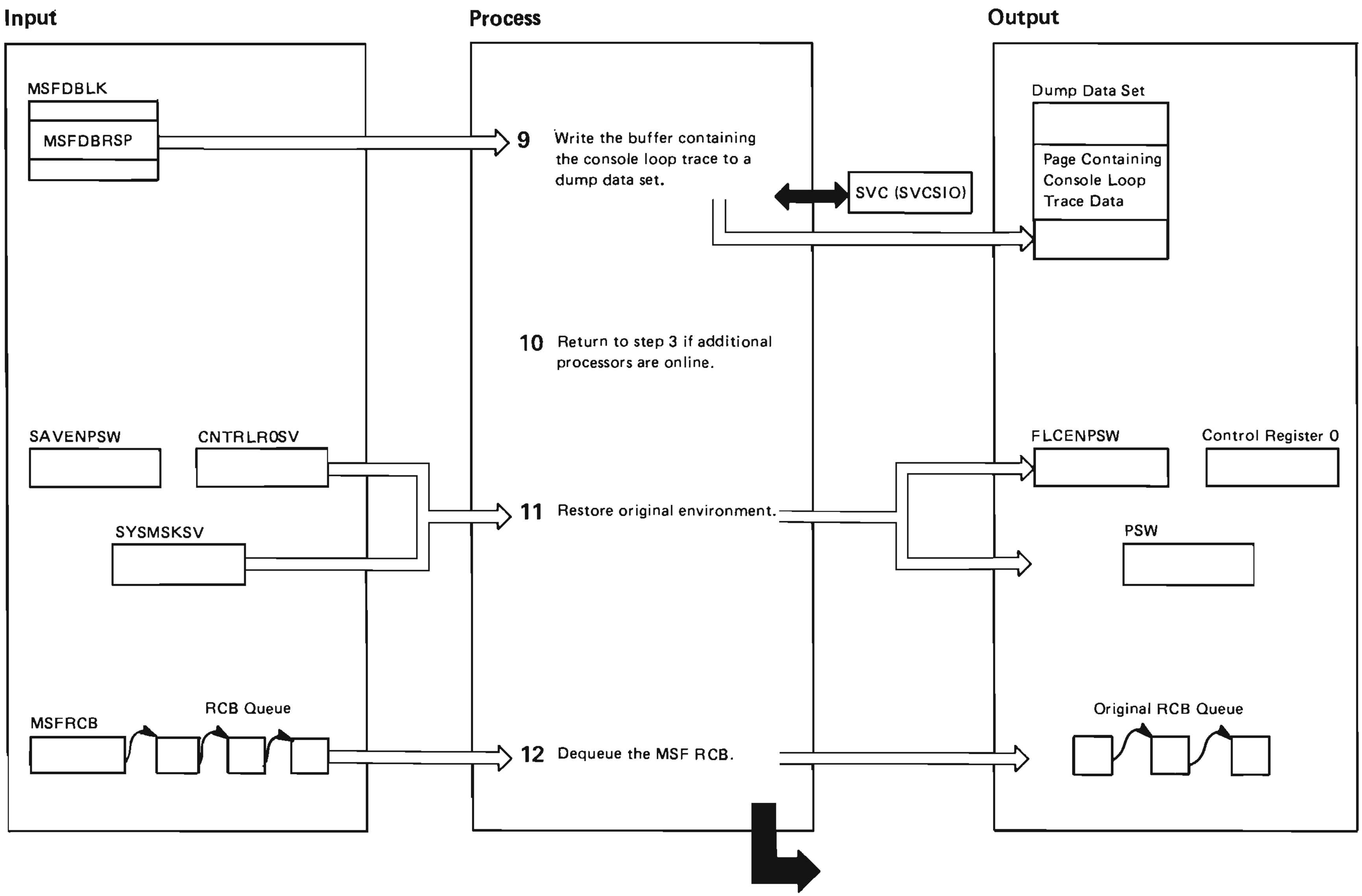


Diagram SADMP-20. AMDSAMSF – Console Loop Trace Buffer Dump (Part 6 of 6)

Extended Description	Module	Label
9 If the data block contains console loop trace data pass control to AMDSASIO via an SVC instruction. AMDSASIO writes the buffer containing the console loop trace to the dump data set.		
10 If more processors are online, AMDSAMSF returns to step 3.		
11 AMDSAMSF restores its environment to allow AMDSADMP to function normally.		
12 AMDSAMSF dequeues the MSF RCB from the RCB queue.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 1 of 16)

Input

S99RBPTR

Process

1 Initialize the DYNALLOC model request block.

2 Call DYNALLOC to allocate GENPRINT.

DYNALLOC

– If the allocation fails, set an error return code and terminate.

Error. Return to the Caller.

– If the allocation is successful, call IO to open GENPRINT.

IO

3 Call IO to open GENPARMS.

IO

– If the open fails, call IONOPEN to write an error message.

IONOPEN

Output

S99RBPTR

S99RB

Register 15

8

IORECA

AMD035I

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 2 of 16)

Extended Description	Model	Label
AMDSAOSG generates a SADMP program that can be IPLed.		
1 AMDSAOSG sets the request block pointer to the request block and initializes the DYNALLOC model request block.		
2 AMDSAOSG calls DYNALLOC to dynamically allocate GENPRINT. If the allocation fails, DYNALLOC sets an error return code of 4 and terminates abnormally. If the allocation is successful, AMDSAOSG calls IO to open GENPRINT.		
3 AMDSAOSG calls IO to open GENPARMS. If the open fails, IO calls IONOPEN to write error message AMD035I.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 3 of 16)

Input

Process

4 Call DYNALLOC to allocate SYSIN.

- If the allocation fails, set an error return code and call IO to write an error message.
- If the allocation is successful, call IO to open SYSIN.
- If the open fails, call IONOPEN to write an error message.

DYNALLOC

IO

IO (OPENOUT, SYSIN)

IONOPEN

Output

Register 15

4

IORECA

AMD063I

IORECA

AMD035I

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 4 of 16)

Extended Description	Module	Label
4 AMDSAOSG calls DYNALLOC to dynamically allocate SYSIN. If the allocation fails, DYNALLOC sets an error return code of 4 and calls IO to write error message AMD063I. If the allocation is successful, AMDSAOSG calls IO to open SYSIN. If the open fails, IO calls IONOPEN to write error message AMD035I.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 5 of 16)

Input

GENPARMS

Process

5 Call IO to generate AMDSADMP macro input and to copy GENPARMS.

IO

6 Call IO to close SYSIN and GENPARMS.

IO

7 Call DYNALLOC to allocate SYSLIB, SYSUT1, and SYSPUNCH.

DYNALLOC

- If the allocation fails, set an error return code, call IO to write an error message, and terminate.

IO

Error. Return to the Caller.

Output

Register 15

4

IORECA

AMD063I

Diagram SADMP-21. AMDSAOSG — One-Step Generation (Part 6 of 16)

Extended Description	Module	Label
5 AMDSAOSG calls IO and generates input for the assembler. Input received from MACVSET1 and MACVSET2 instruct AMDSADMP to invoke AMDSADM2 directly for one-step generation.		
6 AMDSAOSG calls IO to close SYSIN and GENPARMS.		
7 AMDSAOSG calls DYNALLOC to allocate SYSLIB, SYSUT1, and SYSPUNCH. If any of the allocations fail, DYNALLOC sets an error return code of 4, calls IO to write error message AMD063I, and terminates abnormally.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 7 of 16)

Input

Process

8 Call RALSVSP to allocate SYSPRINT.

- If allocation fails, set an error return code, call IO to write an error message, and terminate.

RALSYSP

IO

Error. Return to the Caller.

Output

DYNALLOC

Register 15

4

IORECM

AMD063I

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 8 of 16)

Extended Description	Module	Label
8 AMDSAOSG calls RALSYSP to allocate SYSPRINT for the assembler. RALSYSP calls DYNALLOC. If the allocation fails, DYNALLOC sets an error return code of 4, calls IO to write error message AMD063I, and terminates abnormally.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 9 of 16)

Input

GENPRINT

Assembler Listing

Process

9 Assemble the AMDSACCT and AMDSARDM modules and AMDSAOSG parameter statements.

- If the assembly is successful, call COPYPRNT.
- If the assembly fails, set an error return code, call IO to write an error message, and terminate.

COPYPRNT

IO

Error. Return to the Caller.

10 Call IO to open, read, and close SYSPUNCH.

- If the open fails, call IONOPEN to write an error message.

IO

IONOPEN

11 Call DYNALLOC to allocate IPLTEXT, PGETEXT, IPLDEV, and TRK0TEXT.

- If any of the allocations fail, set an error return code, call IO to write an error message, and terminate.

DYNALLOC

IO

Error. Return to the Caller.

Output

GENPRINT

Assembler Listing

Register 15: 4

IORECA: AMD064I

IPLTYPE

DSFPTEXT

IPLUNIT

IPLSER

IORECA: AMD035I

Register 15: 4

IORECA: AMD063I

Extended Description	Module	Label
9 AMDSAOSG assembles the AMDSACCT and AMDSARDM modules and the AMDSAOSG parameter statements by linking to the assembler. If the assembly is successful, AMDSAOSG calls COPYPRNT to copy the assembler listing to the end of GENPRINT. If the assembly fails, AMDSAOSG sets an error return code of 4, calls IO to write error message AMD064I, and terminates abnormally.		
10 AMDSAOSG calls IO to open SYSPUNCH. If the open fails, IO calls IONOPEN to write error message AMD035I. IO reads the parameter statements from the AMDSADMP output in SYSPUNCH. AMDSAOSG generates the AMDSABLD input parameter, the IPL unit class, the IPL unit type, the IPL serial number, and the ICKDSF input statements. AMDSAOSG calls IO to close SYSPUNCH.		
11 AMDSAOSG calls DYNALLOC to allocate IPLTEXT for AMDSABLD and ICKDSF, to allocate PGETEXT for AMDSABLD, to allocate IPLDEV for AMDSABLD and ICKDSF, and to allocate TRK0TEXT for AMDSABLD and ICKDSF. If any allocation fails, DYNALLOC sets an error return code of 4, calls IO to write error message AMD063I, and terminates abnormally.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 11 of 16)

Input

GENPRINT

Assembler Listing

AMDSABLD SYSPRINT Data Set

Process

12 Call RALSYSP to reallocate SYSPRINT. ↔ RALSYSP ↔ DYNALLOC

- If the allocation fails, set an error return code, call IO to write an error message, and terminate. → Register 15: 4; IO → IORECA: AMD063I

Error. Return to the Caller.

13 Link to AMDSABLD.

- If successful, call COPYPRNT. ↔ COPYPRNT → GENPRINT (Assembler Listing, AMDSABLD SYSPRINT Data Set)

- If AMDSABLD fails, set an error return code, call IO to write an error message, and terminate. → Register 15: 4; IO; IORECA: AMD064I

Error. Return to the Caller.

Output

DYNALLOC

Register 15: 4

IORECA: AMD063I

GENPRINT

Assembler Listing

AMDSABLD SYSPRINT Data Set

Register 15: 4

IORECA: AMD064I

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 12 of 16)

Extended Description	Module	Label
12 AMDSAOSG calls RALSYSP to reallocate SYSPRINT for AMDSABLD. RALSYSP calls DYNALLOC. If the allocation fails, DYNALLOC sets an error return code of 4, calls IO to write error message AMD063I, and terminates abnormally.		
13 AMDSAOSG links to AMDSABLD. If the IPL unit is tape, the link initializes the SADMP residence volume. If the IPL unit is DASD, AMDSAOSG uses ICKDSF to initialize track 0. If the link is successful, AMDSAOSG calls COPYPRNT to copy the AMDSABLD printed output to the end of GENPRINT. If AMDSABLD fails, AMDSAOSG sets an error return code of 4, calls IO to write error message AMD064I, and terminates abnormally.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 13 of 16)

Input

IPLTYPE

=DASD

Process

14 For a DASD IPL:

a. Call IO to open, read, and close SYSIN. → IO

– If the open fails, call IONOPEN to write an error message. → IONOPEN → IORECA: AMD035I

b. Call RALSYSP to reallocate SYSPRINT. → RALSYSP ↔ DYNALLOC

– If allocation fails, set an error return code, call IO to write an error message, and terminate. → Register 15: 4; IO → IORECA: AMD063I

Error. Return to the Caller.

Output

IORECA: AMD035I

DYNALLOC

Register 15: 4

IORECA: AMD063I

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 14 of 16)

Extended Description	Module	Label
14 a. For DASD IPL units, AMDSAOSG calls IO to open SYSIN. If the open fails, IO calls IONOPEN to write error message AMD035I. If the open is successful, AMDSAOSG calls IO to copy the ICKDSF parameters into SYSIN, and close SYSIN.		
b. AMDSAOSG calls RALSYSP to reallocate SYSPRINT for DSF. RALSYSP calls DYNALLOC. If the allocation fails, DYNALLOC sets an error return code of 4, calls IO to write error message AMD063I, and terminates abnormally.		

Diagram SADMP-21. AMDSAOSG – One-Step Generation (Part 15 of 16)

Input

GENPRINT

- Assembler Listing
- AMDSABLD SYSPRINT Data Set

ICKDSF SYSPRINT Data Set

Process

c. Link to ICKDSF.

- If successful, call COPYDSFP.
- If ICKDSF fails, set an error return code, call IO to write an error message, and terminate.

COPYDSFP

IO

Error. Return to the Caller.

15 Call IO to write a completion message and to set a return code.

IO

16 Close and deallocate files, and return.

Return to Caller.

Output

GENPRINT

- Assembler Listing
- AMDSABLD SYSPRINT
- ICKDSF SYSPRINT

Register 15: 4

IORECA: AMD064I

IORECA: AMD062I

Register 15: 0

Diagram SADMP-21. AMDSAOSG — One-Step Generation (Part 16 of 16)

Extended Description	Module	Label
14 c. AMDSAOSG links to ICKSDF. If the link is successful, AMDSAOSG calls COPYDSFP to copy the printed output from ICKDSF to the end of GENPRINT. If ICKDSF fails, AMDSAOSG sets an error return code of 4, calls IO to write error message AMD0641, and terminates abnormally.		
15 AMDSAOSG calls IO to write successful completion message AMD062I, and sets a return code of 0.		
16 AMDSAOSG closes and deallocates the files and returns.		

Diagram SADMP-22. AMDSAPGE – Virtual Storage Dump Initialization and Control (Part 1 of 8)

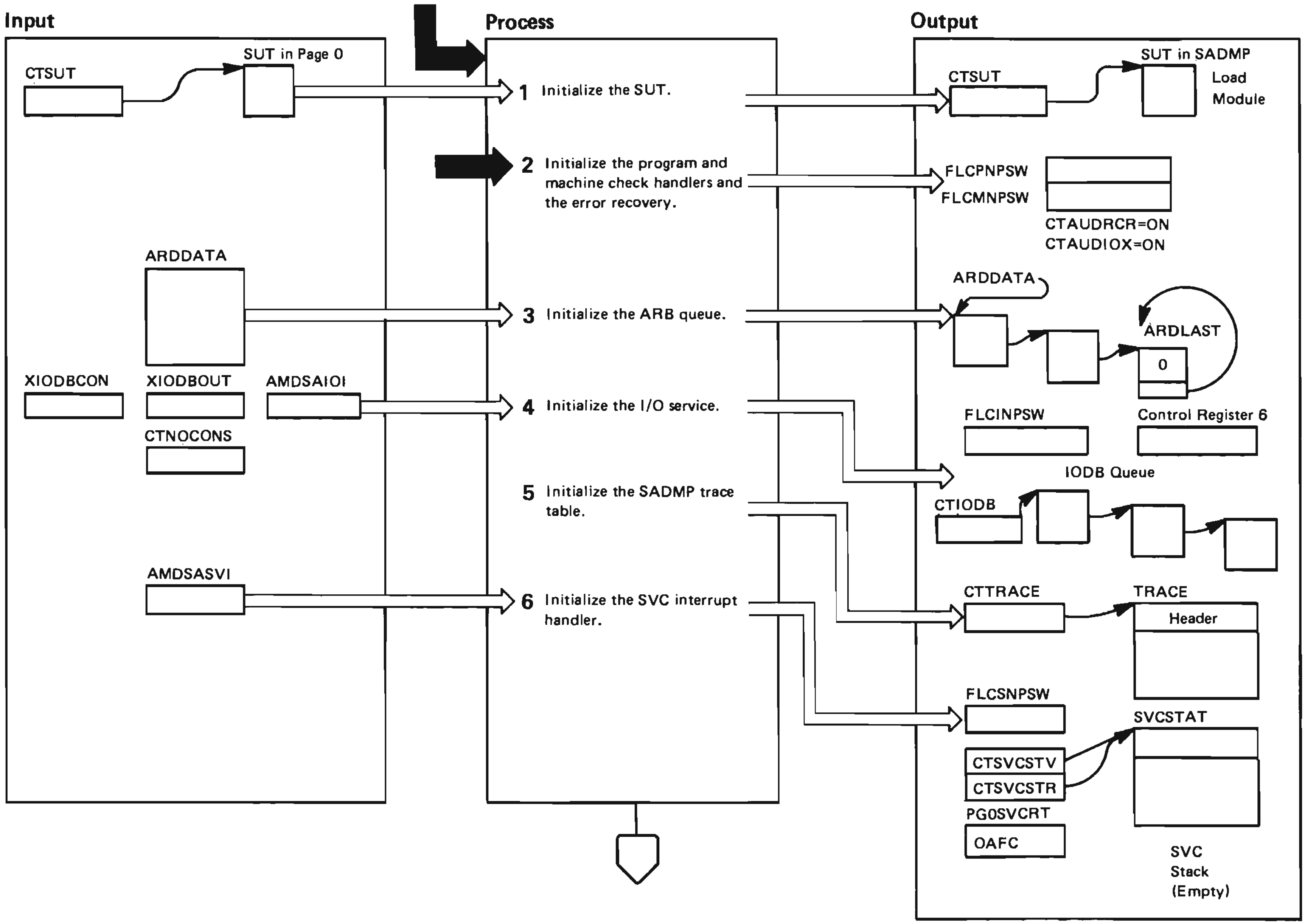
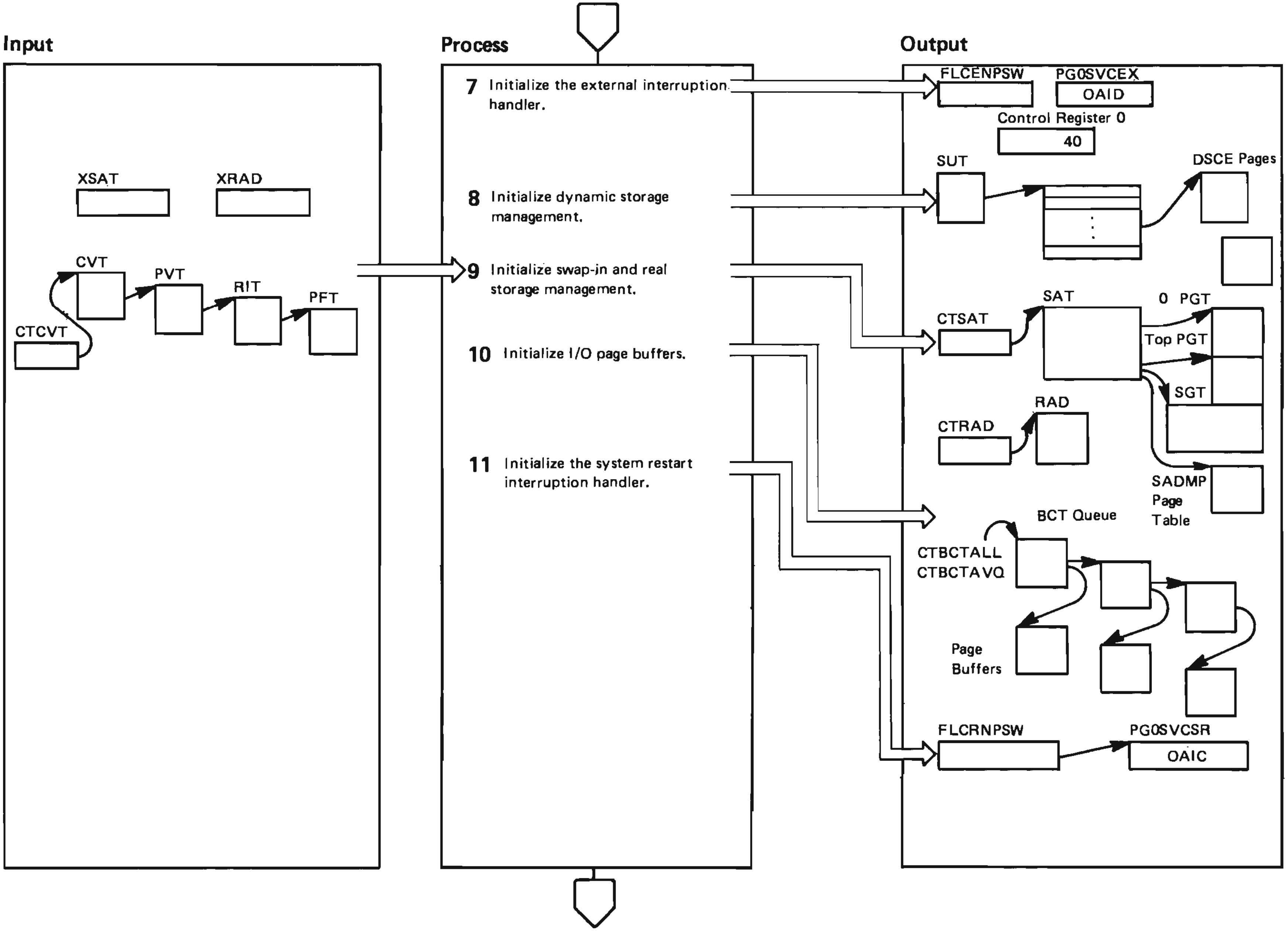


Diagram SADMP-22. AMDSAPGE — Virtual Storage Dump Initialization and Control (Part 2 of 8)

Extended Description	Module	Label
1 Entry point AMDSAPGE in the CSECT AMDSAPGE initializes the virtual dump program and controls its execution and termination.		
2 The program new PSW points to module AMDSAPGI. The machine check new PSW is a disabled wait PSW with code X'370000'. After initialization, any successful system restart causes AMDSAPGE to reinitialize SADMP beginning with this step. System restart is not permitted during SADMP initialization. The restart new PSW points to AMDSARST, which reloads the restart old PSW.		
3 AMDSAPGE sets the address range block (ARB) queue header in ARDDATA to zero. The ARB queue header is the last and only ARB in the queue.		
4 The I/O new PSW points to AMDSAIOI. AMDSAPGE builds the IODBs and their related control blocks using the information found in the CCT. AMDSAPGE disables the IPL subchannel and sets control register 6 to permit interruptions from the output tape but not the console.		
5 SADMP uses the second page of the AMDSAPGE load module for its trace table. AMDSAPGE clears the page, puts 'TRCT' into the header, and sets the current index to zero.		
6 The first virtual page after the AMDSAPGE load module is used for the SVC stack. AMDSAPGE clears the page, puts 'SVCS' into the header, and sets the current index to zero.		

Diagram SADMP-22. AMDSAPGE – Virtual Storage Dump Initialization and Control (Part 3 of 8)



Extended Description	Module	Label
7 The new external interrupt PSW points to an SVC in page 0 so external interrupts are converted into SVC interrupts. Only console interrupts are enabled.		
8 The next two unallocated pages are set aside for the dynamic storage control elements (DSCE). The next 16 pages, following the DSCE, are set aside for SADMP dynamic storage. Each of the 16 pages is mapped by one DSCE. These DSCEs are placed on the DSCE in-use queue. The rest are placed on the DSCE available queue.		
9 The swap address table (SAT) contains the real and virtual addresses of the following areas that SADMP uses to swap-in an address space: the page tables for segment 0 and the top segment, the segment table (allocated from previously unallocated SADMP storage), and SADMP's own page table. The CCT points to the SAT and the RAD.		
10 SADMP storage that is still unallocated is used for SADMP page buffers and the BCTs that support them. The BCTs are queued together in the BCT available queue or the queue of all BCTs.		
11 The restart new PSW points to an SVC in page 0 by which a system restart is converted into an SVC interrupt.		

Diagram SADMP-22. AMDSAPGE – Virtual Storage Dump Initialization and Control (Part 5 of 8)

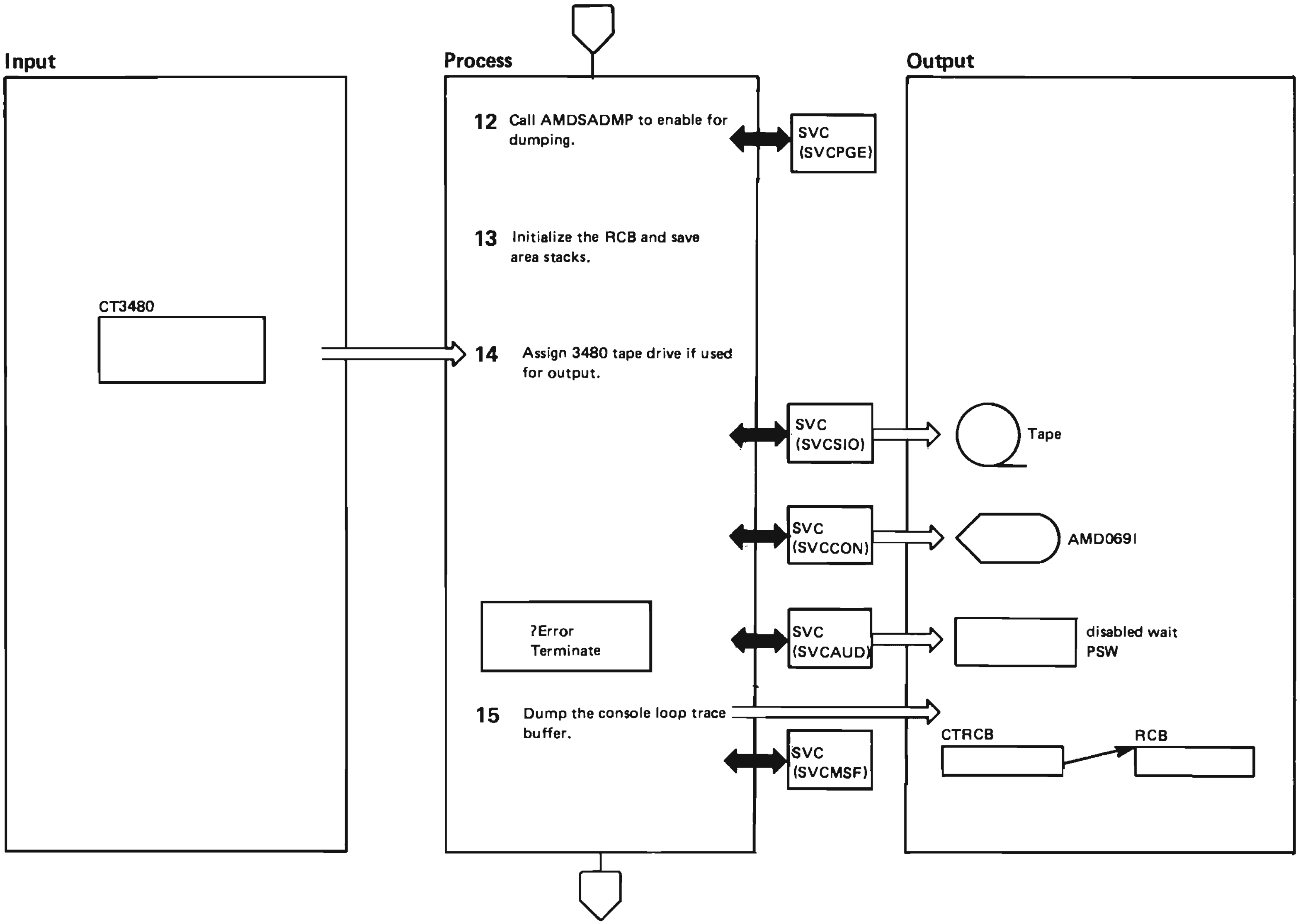


Diagram SADMP-22. AMDSAPGE – Virtual Storage Dump Initialization and Control (Part 6 of 8)

Extended Description	Module	Label
12 AMDSAPGE becomes enabled for I/O and external interrupts. This SVC puts the first entry into the SVC stack.		
13 The save area pointer is set to zero. The AMDSAPGE RCB defines error recovery for the remainder of AMDSAPGE and serves to end error recovery percolation since it is the last RCB on the RCB queue. The AMDSAPGE RCB causes the virtual dump to terminate with an error code that indicates the segment of AMDSAPGE last in control.		
14 If the output device is a 3480 tape drive, AMDSAPGE attempts to assign the tape drive. If the assign command fails to assign path groups, AMDSAPGE issues message AMD069I and terminates SADMP.		
15 AMDSAPGE calls AMDSAMSF (via an SVC macro) to dump the console loop trace buffer.		

Diagram SADMP-22. AMDSAPGE – Virtual Storage Dump Initialization and Control (Part 7 of 8)

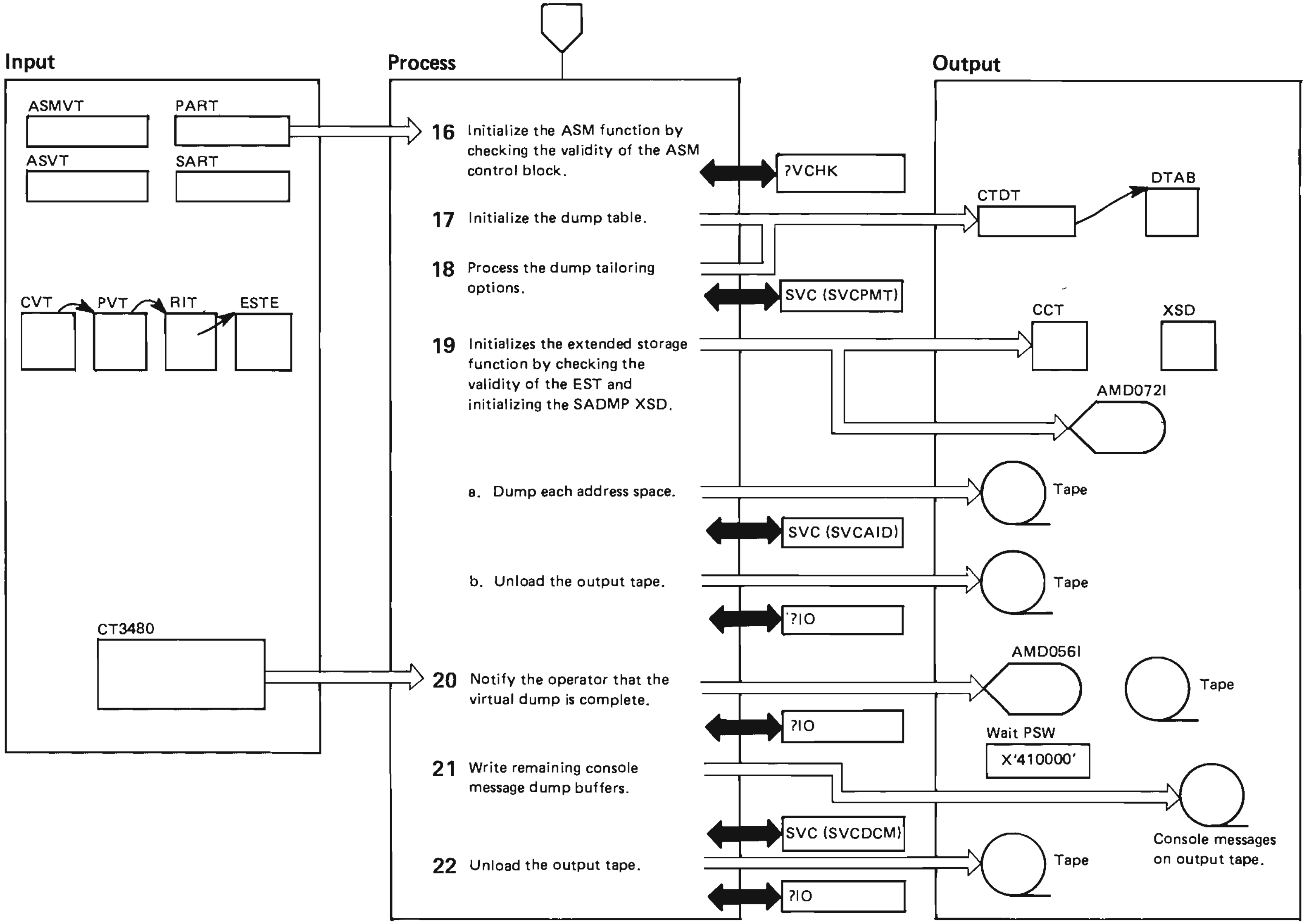


Diagram SADMP-22. AMDSAPGE — Virtual Storage Dump Initialization and Control (Part 8 of 8)

Extended Description	Module	Label
16 AMDSAPGE checks the validity of the ASM control blocks, the ASMVT, the ASVT, and the PART. If the ASVT control blocks are invalid, AMDSAPGE cannot perform paging operations, so AMDSAPGE terminates. If the ASMVT or PART is invalid, AMDSAPGE continues processing; however, it does not dump virtual storage from page data sets. AMDSAPGE also checks the validity of the SART. If the SART is invalid, AMDSAPGE continues processing; however, it does not dump swapped-out address spaces.		
17 AMDSAPGE obtains storage for the dump table, and initializes the dump table to request dumping of subpools 229, 230, 236, and 237 in all address spaces.		
18 AMDSAPGE calls AMDSAPMT (via an SVC macro) to fill in the dump table using the dump tailoring options.		
19 AMDSAPGE checks the validity of the MVS extended storage table (EST). If it is valid, AMDSAPGE initializes the SADMP extended storage descriptor (XSD). If it is not valid, AMDSAPGE writes message AMD072I.		
20 AMDSAPGE calls AMDSAAID (via an SVC macro) to dump each address space.		
21 AMDSAPGE issues message AMD056I.		
22 AMDSAPGE calls AMDSADCM to write any remaining console message dump buffers to the output tape.		
23 AMDSAPGE waits for the output tape to finish, then unloads the tape. If the output is to a 3480 drive, AMDSAPGE writes a display message to the tape drive indicating that SADMP has completed processing. Also, if the output device is a 3480 tape drive, AMDSAPGE attempts to unassign the tape drive. Then, in all cases, AMDSAPGE loads a wait state PSW with an address of X'410000'. This notifies the operator that the virtual storage dump completed successfully.		

Diagram SADMP-23. AMDSAPGI – Virtual Storage Dump Program Check Handler (Part 1 of 4)

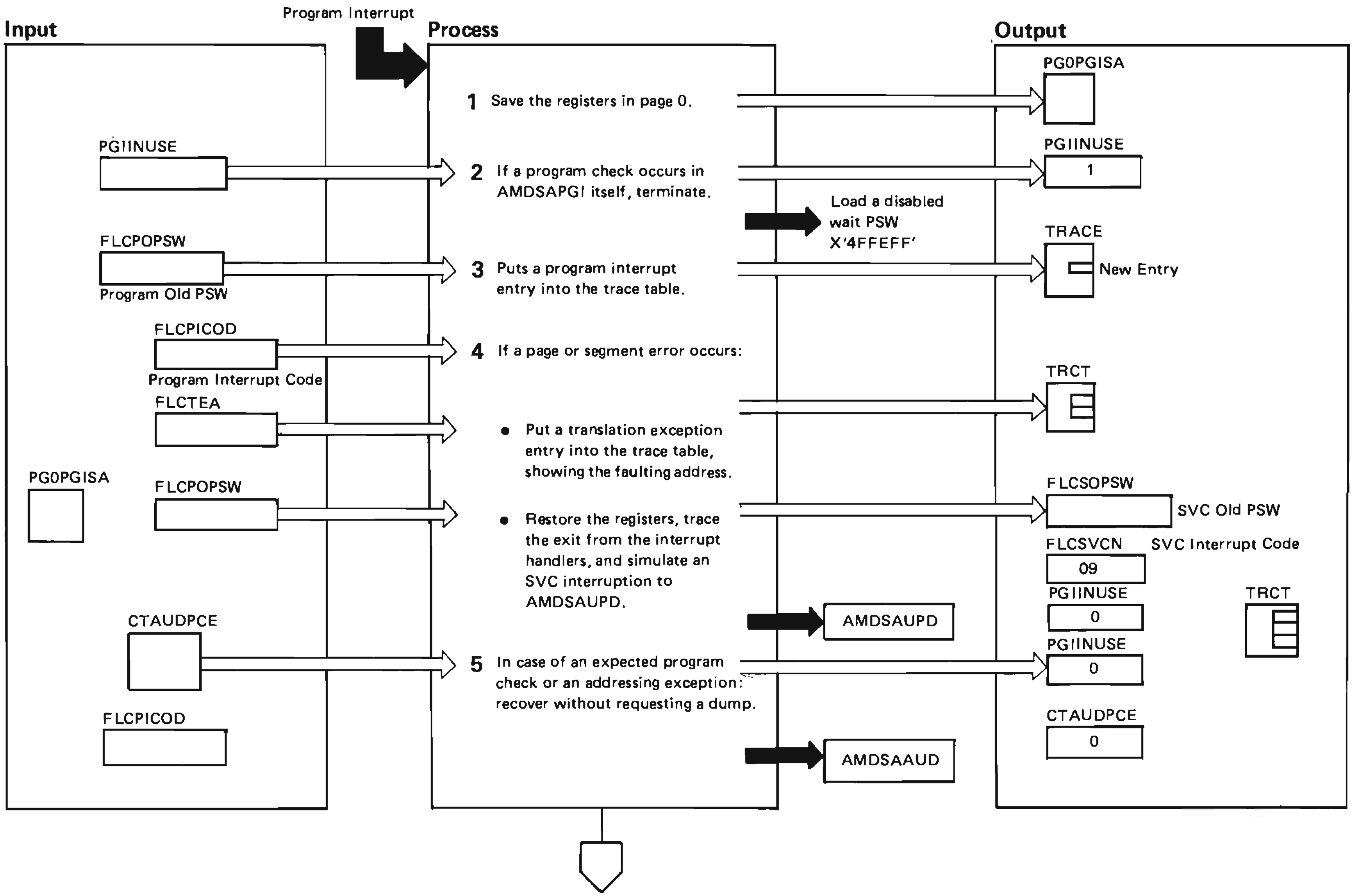


Diagram SADMP-23. AMDSAPGI — Virtual Storage Dump Program Check Handler (Part 2 of 4)

Extended Description	Module	Label

Several types of program checks can occur. Program checks in AMDSAPGI are severe errors causing immediate termination. Translation exceptions are normal, and handled by converting the program check into an apparent SVC interrupt. Expected program checks or addressing exceptions are normal. AMDSAPGI treats other program checks as SADMP errors that require cautious recovery and possibly abnormal termination.

1 AMDSAPGI copies the entry registers into page 0 (PG0PGISA).

2 If the program check is recursive (PGIINUSE = 1), AMDSAPGI terminates immediately by loading a disabled wait. If it is not recursive (PGIINUSE = 0), AMDSAPGI protects against recursion by setting PGIINUSE = 1.

3 AMDSAPGI records the program check (program old PSW and trace ID) in the SADMP trace table (CTTRACE).

4 If the interrupt is a translation exception (segment or page fault, FLCPICOD = X'0010' or X'0011'), AMDSAPGI also records the faulting address (FLCTEA) in the trace table by putting it into a dummy PSW. AMDSAPGI simulates an SVC call to AMDSAUPD by the copying program-check old PSW (FLCPOPSW) to the SVC new PSW (FLCSOPSW), placing the SVC number for AMDSAUPD into the SVC interrupt code (FLCSVCN), reloading the entry registers, and loading the SVC new PSW (FLCSNPSW). AMDSAPGI also traces the exit from AMDSAPGI.

5 If the program check is an addressing exception (PIC = 5) or the program-check was expected, AMDSAPGI sets the recursion lock PGIINUSE and the program-check-expected flag CTAUDPCE to 0, and calls AMDSAAUD to give control to the top RCB exit.

Diagram SADMP-23. AMDSAPGI – Virtual Storage Dump Program Check Handler (Part 3 of 4)

Input	Process	Output
	6 If a page or segment fault does not occur and there is a program check in the DAT-on module, trace call to AMDSAAUD, dump and recover. → AMDSAAUD	

Diagram SADMP-23. AMDSAPGI — Virtual Storage Dump Program Check Handler (Part 4 of 4)

Extended Description	Module	Label
6 For all other program checks, AMDSAPGI calls for a dump (CTAUDDMP = ON), asks for recovery (CTAUDRCV = ON), and calls AMDSAAUD.		

Diagram SADMP-24. AMDSAPMT – Interactive Prompting for Dump Options (Part 1 of 2)

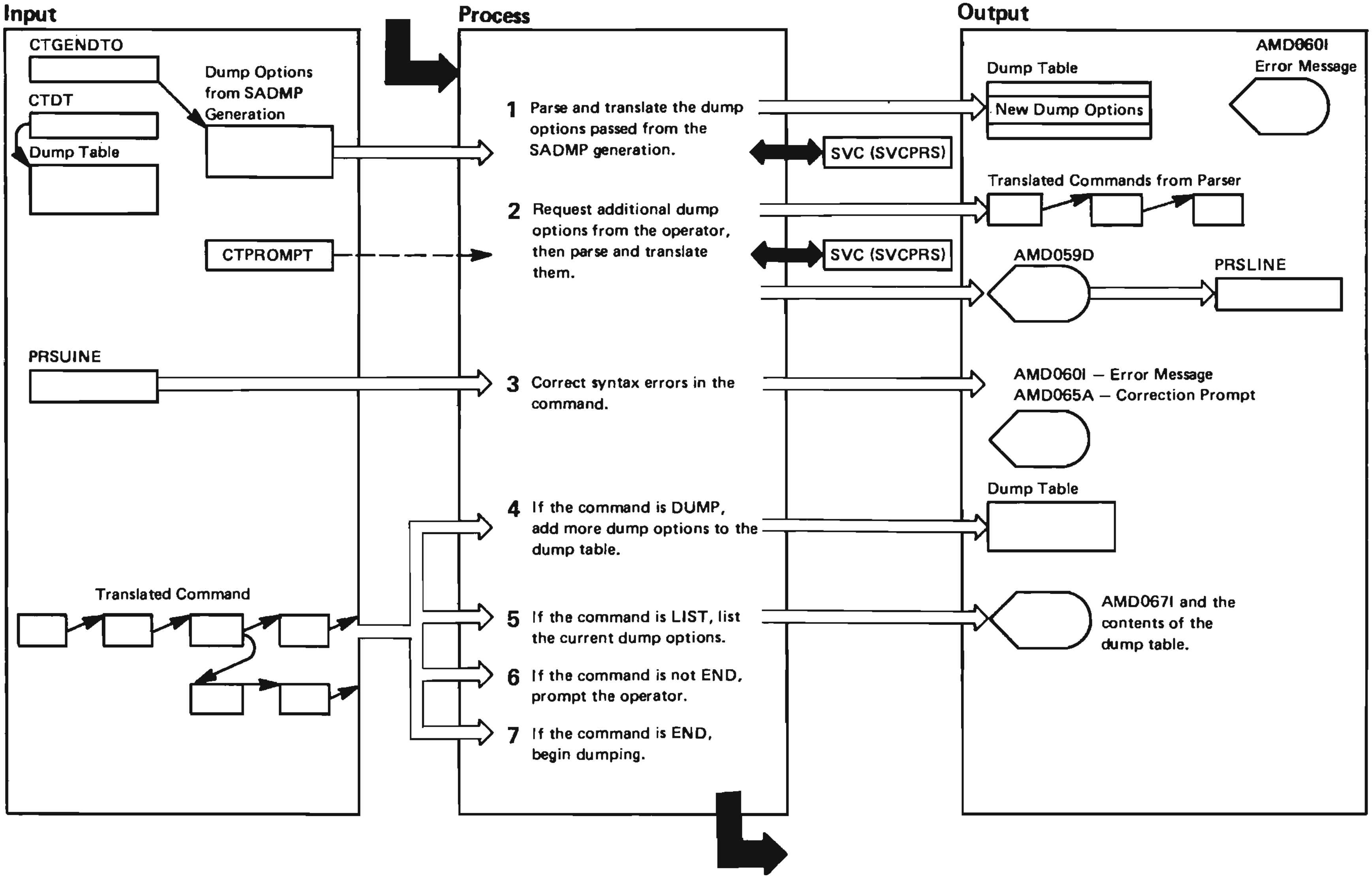


Diagram SADMP-24. AMDSAPMT – Interactive Prompting for Dump Options (Part 2 of 2)

Extended Description	Module	Label
1 If CTGENDTO is nonzero, CTGENDTO points to the dump options text entered during SADMP generation. AMDSAPMT calls AMDSAPRS to translate it into a dump option set (pointed to by PTHPTR), then compresses the dump option set into a dump table that the CTDT points to. If CTGENDTO is zero, this indicates that AMDSAPMT has no dump options next to translate.		
2 If CTPROMPT=ON, AMDSAPMT writes message AMD059D to prompt the operator for more dump options. AMDSAPMT reads the reply from the console into PRSLINE, then passes PRSLINE to AMDSAPRS to have it parsed and translated.		
3 If the return code from AMDSAPRS is nonzero, an error exists in PRSLINE. AMDSAPMT writes message AMD060I to show the syntax error. AMDSAPMT repeats PRSLINE and underlines the text that is in error. AMDSAPMT writes message AMD065A to ask the operator to correct the error. AMDSAPMT reads the operator's correction and inserts it into PRSLINE in place of the asterisked text. AMDSAPMT continues until PRSLINE is free of syntax errors. A return code of 0 from AMDSAPRS indicates no more errors in PRSLINE.		
4 AMDSAPMT calls internal subroutine COMPRESS to compress the dump option set into a list, which it appends to the end of the dump table that the CTDT points to.		
5 AMDSAPMT calls internal subroutine LISTDT to write message AMD067I to the console, then expands the dump table entries into an EBCDIC form that agrees with the dump options syntax.		
6,7 If the command is END, AMDSAPMT exits; otherwise AMDSAPMT reprompts the operator.		

Diagram SADMP-25. AMDSAPRS – SADMP Parsing and Translation (Part 1 of 2)

Input

Parameter List

INPLINE
OUTPTHP
OUTERBEG
OUTEREND

TKTABLE

Process

1 Convert the input text into a table of tokens.

2 Parse and translate the token table into a parse-translation header (PTH) and possibly a dump option set.

3 If the syntax of the input is correct, return the PTH to the caller.

4 If the syntax of the input is in error, return the bounds of the first token in error to the caller.

Output

TKTABLE

PTH

DOS

PTH

OUTERBEG

OUTEREND

Diagram SADMP-25. AMDSAPRS – SADMP Parsing and Translation (Part 2 of 2)

Extended Description	Module	Label
1 Subroutine LEXSCAN converts the input text (INPLINE) into a token table (TKTABLE) by eliminating blanks and replacing keywords with 1-byte symbols. LEXSCAN also does syntax checking.		
2 Subroutine PARSE uses a stack (PSTACK) and a BNF grammar to check the syntax and translate the token table into a dump options set (DOS) pointed to by a PTH. If there was a syntax error, PARSE indicates which token was the first to be detected in error.		
3,4 The returned pointer to the PTH is OUTPTHP. The bounds of the token in error, as they appear in the input text, are OUTERBEG and OUTEREND.		

Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 1 of 12)

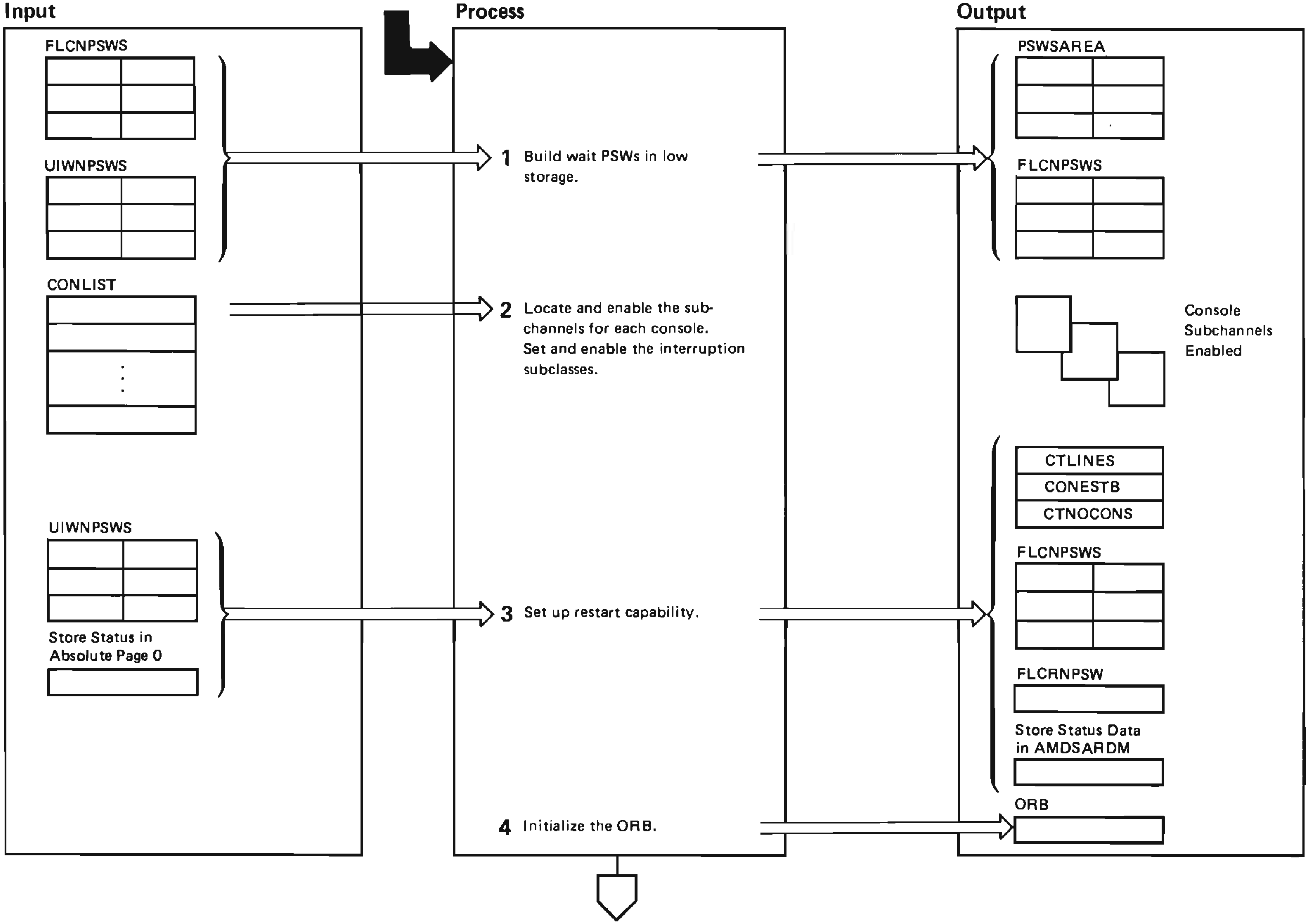


Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 2 of 12)

Extended Description	Module	Label
1 AMDSARDM saves the caller's new PSWs and replaces them with wait PSWs to field all unexpected interrupts.		
2 AMDSARDM locates and enables the subchannels for each of the consoles in the console list. For each console in the list, AMDSARDM sets the interrupt subclass.		
3 AMDSARDM sets on appropriate flags so that the real storage dump program can be retried at this point. AMDSARDM modifies the restart new PSW to pass control to this point is a restart interrupt occurs during real storage dump processing. AMDSARDM saves the store status information for the IPL processor so that the information remains valid for any further restart processing.		
4 AMDSARDM initializes the operation request block (ORB).		

Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 3 of 12)

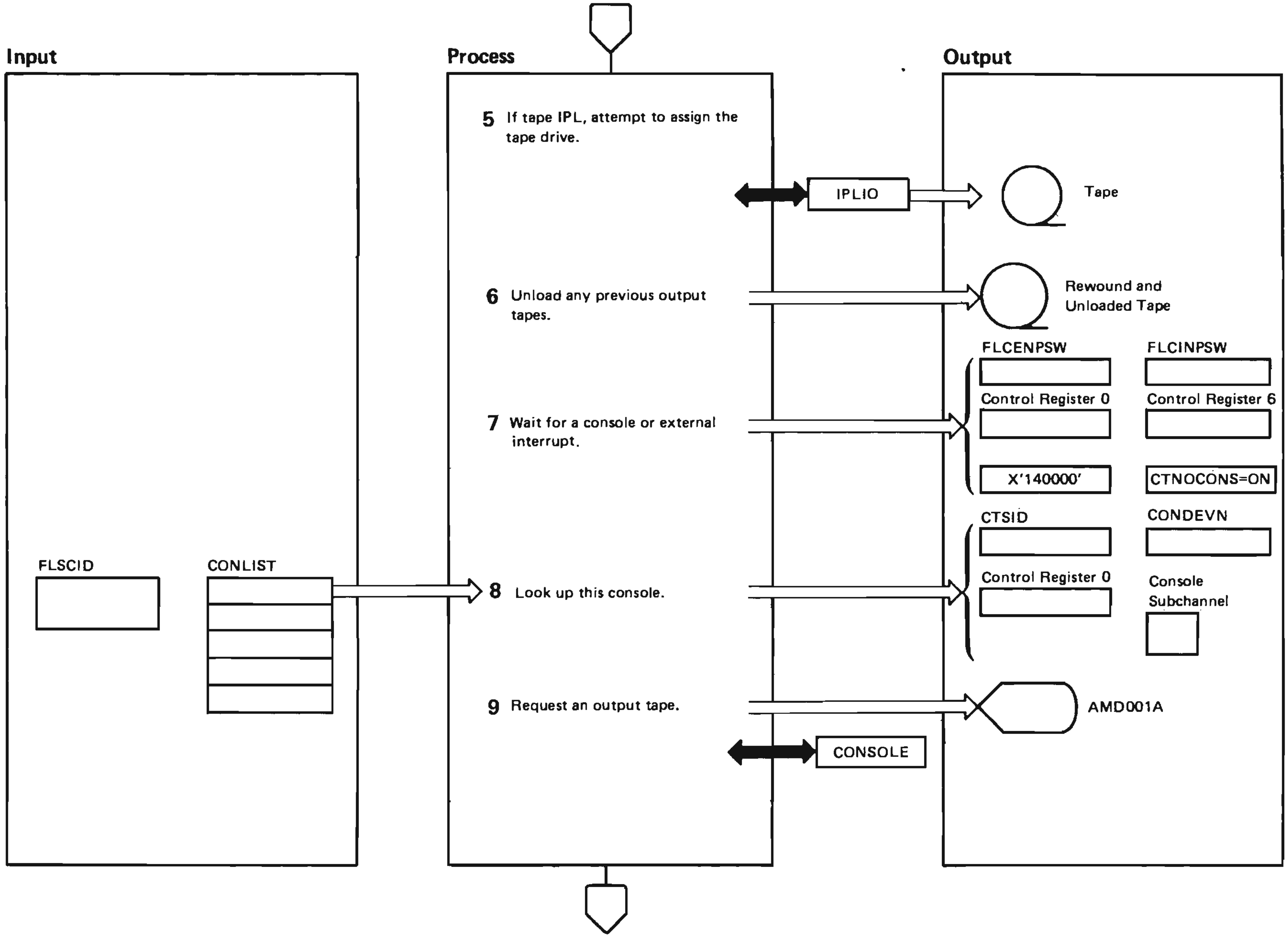
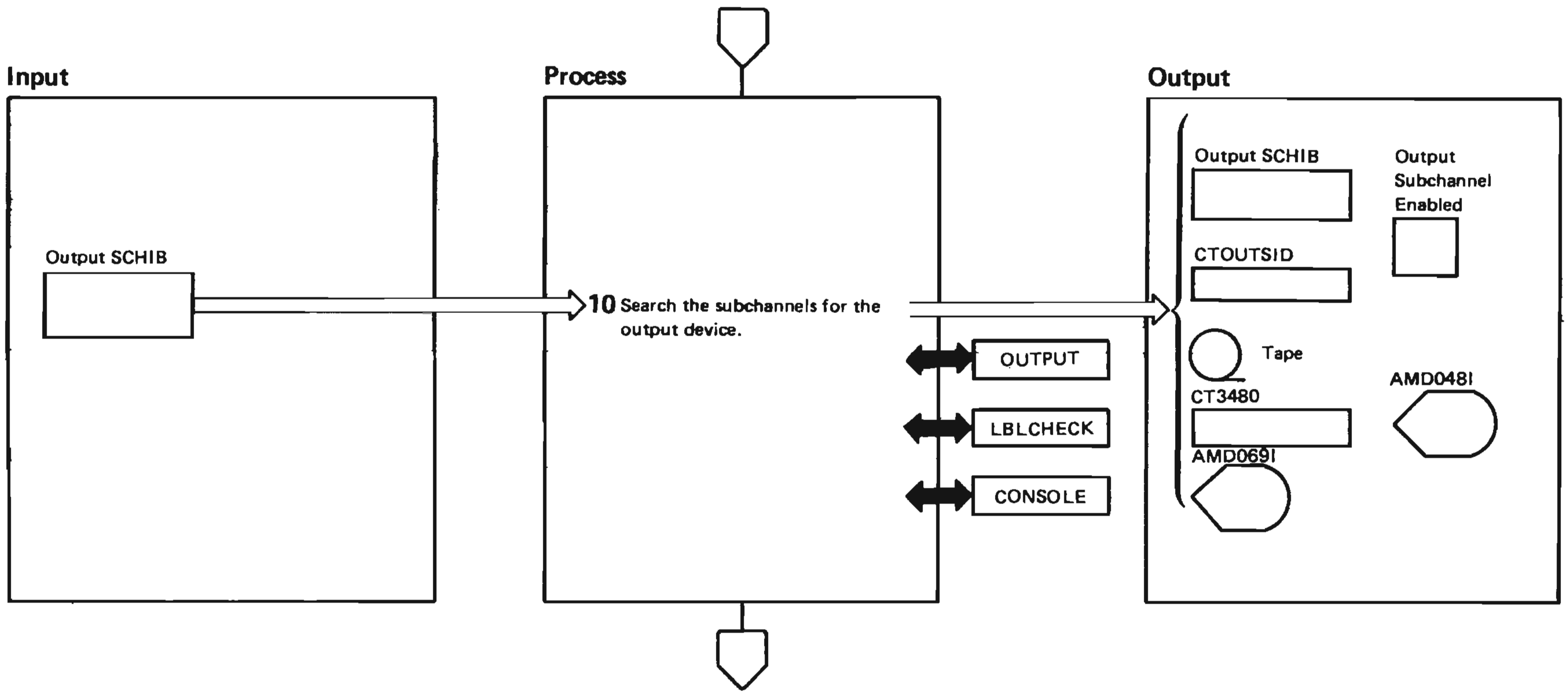


Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 4 of 12)

Extended Description	Module	Label
5 If IPLed from a tape drive, AMDSARDM attempts to assign the tape drive. If the assign fails, AMDSARDM ignores the error because it is too late for SADMP to use another IPL device.		
6 If an output tape was previously set up, AMDSARDM writes an end-of-file mark and unloads the tape.		
7 AMDSARDM sets up the external new PSW and the I/O interrupt new PSW to intercept interrupts. AMDSARDM enables for external signal interrupts and attention interrupt by setting bits in control registers 0 and 6. AMDSARDM loads a wait state PSW to wait for either interrupt. If AMDSARDM receives an external signal interrupt, AMDSARDM turns on the CTNOCONS bit, and SADMP continues processing and does not use a console.		
8 If an attention interrupt occurs, AMDSARDM searches the console list for the appropriate console, and prepares that console for use.		
9 AMDSARDM issues message AMD001A to request the device number of the output tape. If no console is available, AMDSARDM uses the output tape device number that was specified at generation time.		

Diagram SADMP-26. AMDSARDM — Real Storage Dump (Part 5 of 12)



Extended Description	Module	Label
10 AMDSARDM converts the device number to a subchannel identifier, and tests the validity of the subchannel ID. If the subchannel ID is valid, and the device is not already in use by SADMP, AMDSARDM checks if the output device is operational. If the device is operational, AMDSARDM reads the label on the tape. If AMDSARDM accepts the label, or if there is no label, AMDSARDM continues using this tape as the output device.		
If output is to a tape drive, AMDSARDM attempts to assign the tape drive. If the assign command is rejected, the output device is not a 3480 tape drive. If the assign command is accepted, but it fails to assign path groups, AMDSARDM issues message AMD069I, rejecting the tape drive.		
If the output device is a 3480 tape drive, flag CT3480 is set to indicate this. AMDSARDM writes a display message to the tape drive indicating that a tape should be mounted.		
If the tape drive is not a 3480 tape drive or it is and has been assigned, AMDSARDM reads the label on the tape. If AMDSARDM accepts the label, or if there is no label, AMDSARDM continues using this tape as the output device.		
If any of these tests fail, AMDSARDM issues message AMD048I to prompt the operator for a new device number. Then AMDSARDM returns control to step 8.		

Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 7 of 12)

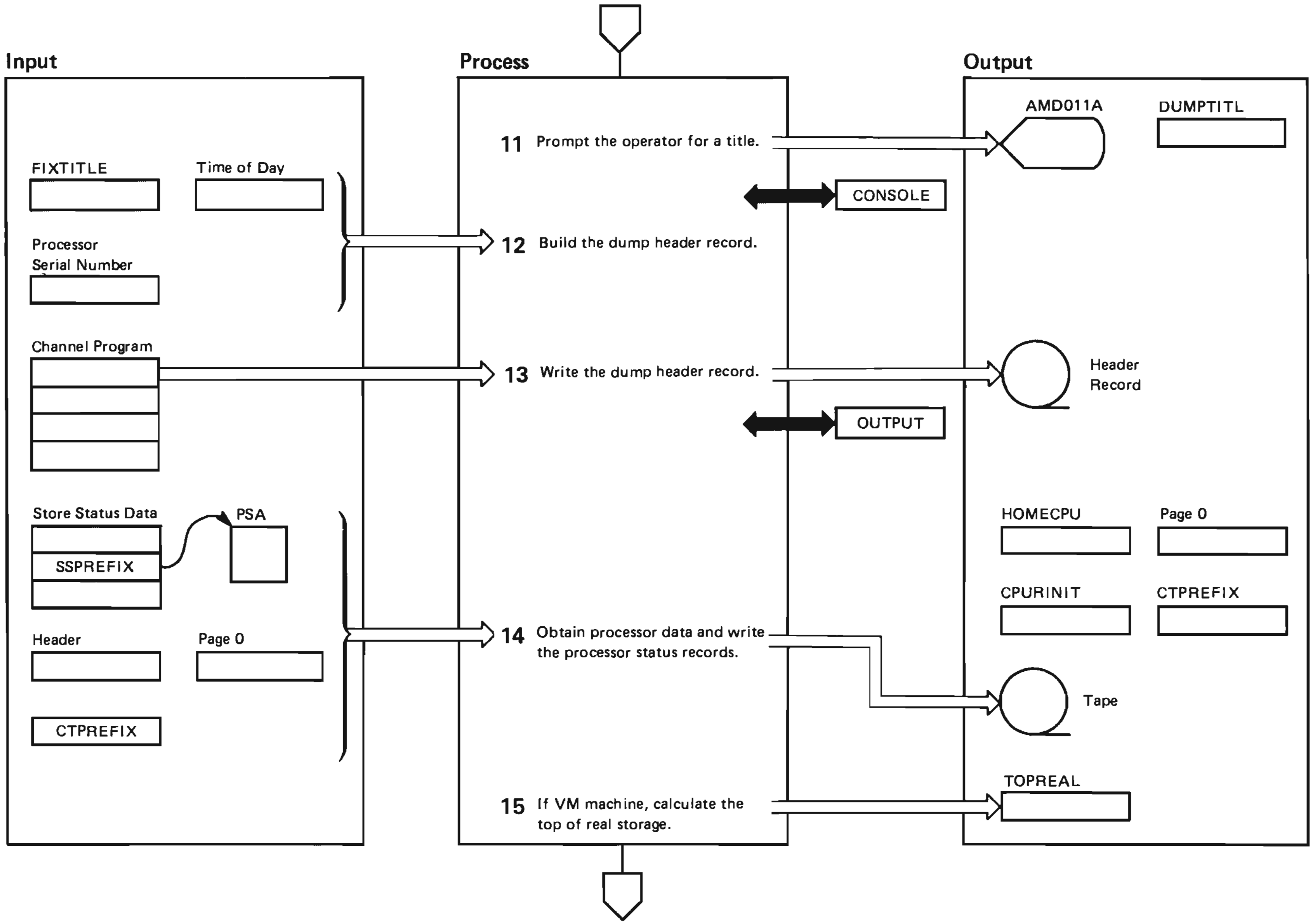


Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 8 of 12)

Extended Description	Module	Label

11 AMDSARDM issues message AMD011A to request a dump title from the operator.

12 AMDSARDM completes the dump header record, using the dump title.

13 AMDSARDM writes the dump header record to the output tape.

14 AMDSARDM obtains the address of the processor that is executing SADMP. If the prefix value from the original store status points to a valid PSA, AMDSARDM saves the PSA. AMDSARDM builds a processor status record for the processor that is executing SADMP, and writes the record to the output tape.

AMDSARDM issues a store status command to each online processor. For each processor, if the prefix value for the store status points to a valid PSA, AMDSARDM saves the PSA. AMDSARDM builds a processor status record for each online processor and writes the records to the output tape.

15 If SADMP is executing on a VM machine, AMDSARDM interfaces with the service processor via a SCP INFO command. From the data that the SCP INFO command returns, AMDSARDM calculates the top of real storage and saves the value in TOPREAL.

Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 9 of 12)

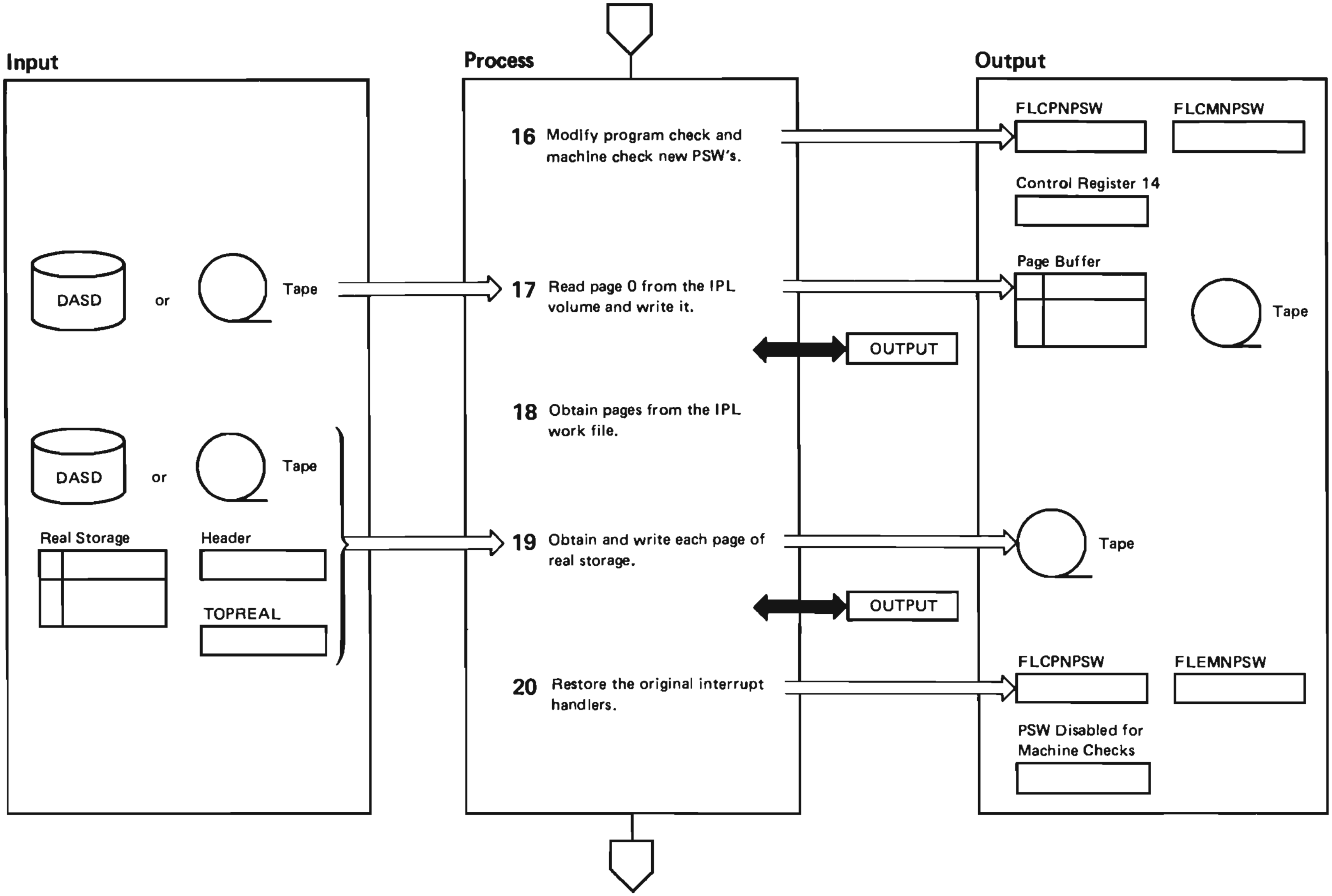


Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 10 of 12)

Extended Description	Module	Label
16 AMDSARDM sets up the machine check new PSW and the program check new PSW to intercept these interrupts if storage errors occur while accessing or dumping real storage. AMDSARDM enables for machine checks by setting bits in control register 14.		
17 AMDSARDM reads page 0 from the IPL volume workfile, and dumps the contents of page 0 to the output tape.		
18 AMDSARDM obtains the pages saved on the IPL volume workfile by AMDSAIPL. AMDSARDM dumps these pages to the output tape.		
19 AMDSARDM obtains each page of real storage and its storage key, and dumps these to the output tape. If SADMP is executing on a VM machine, AMDSARDM uses the value in TOPREAL to determine the highest page to dump. If SADMP is not executing on a VM machine, AMDSARDM attempts to dump all pages to the architectural limit. If a page is not there, the program check handler gets control. The program check handler passes control back to this step so that AMDSARDM can attempt to dump the next page.		
20 AMDSARDM restores the original program check and machine check interrupt environment.		

Diagram SADMP-26. AMDSARDM – Real Storage Dump (Part 11 of 12)

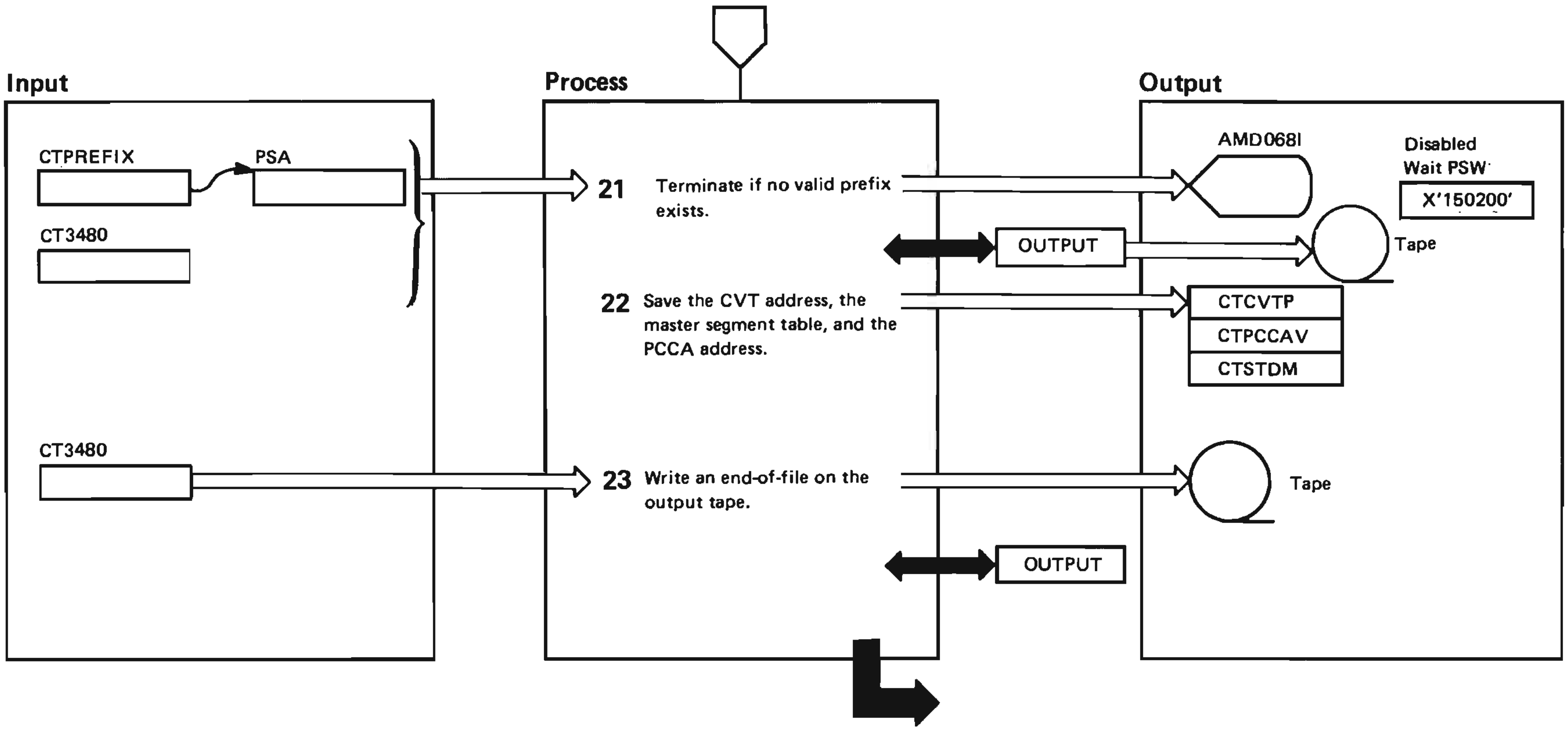


Diagram SADMP-26. AMDSARDM — Real Storage Dump (Part 12 of 12)

Extended Description	Module	Label
21 If the prefix address is not valid, AMDSARDM issues message AMD068I to inform the operator that virtual storage cannot be dumped due to an invalid prefix address. AMDSARDM then loads a disabled wait state PSW. If output is to a 3480 tape drive, AMDSARDM writes a display message to the tape drive indicating that SADMP is complete and attempts to unassign the tape drive. AMDSARDM the loads a disabled wait state PSW.		
22 AMDSARDM saves the addresses of the CVT and the PCCA, and saves the master segment table designation. AMDSADIP, the virtual storage dump program initialization module, later uses this information.		
23 AMDSARDM writes an end-of-file indicator on the output tape so that if the virtual storage dump program cannot be loaded, the tape is complete. If the output is to a 3480 tape drive, AMDSARDM attempts to unassign the tape drive.		

Diagram SADMP-27. CONSOLE – Subroutine of AMDSARDM – Read from and Write to the Operator Console (Part 1 of 4)

Input

CTNOCONS

CTCONSID

Register 6 → Message Parameter

CTLINES

MPFCONT

MPLNCNT

Process

1 If no console is available, return.

2 Set up for console I/O.

3 Compute the position of the message on the screen.

4 If message fills the screen, erase it.

5 Set up the channel program.

6 Perform the write.

– If an error occurs, set a return code.

DUMPSIO

Output

Register 15: 4

Control Register 6: Enabled for Console

0 Old Value Saved

Register 1: Console SID

CTLINES: Beginning of Message

Register 7: End of Message

WORKREG3

AMD029D

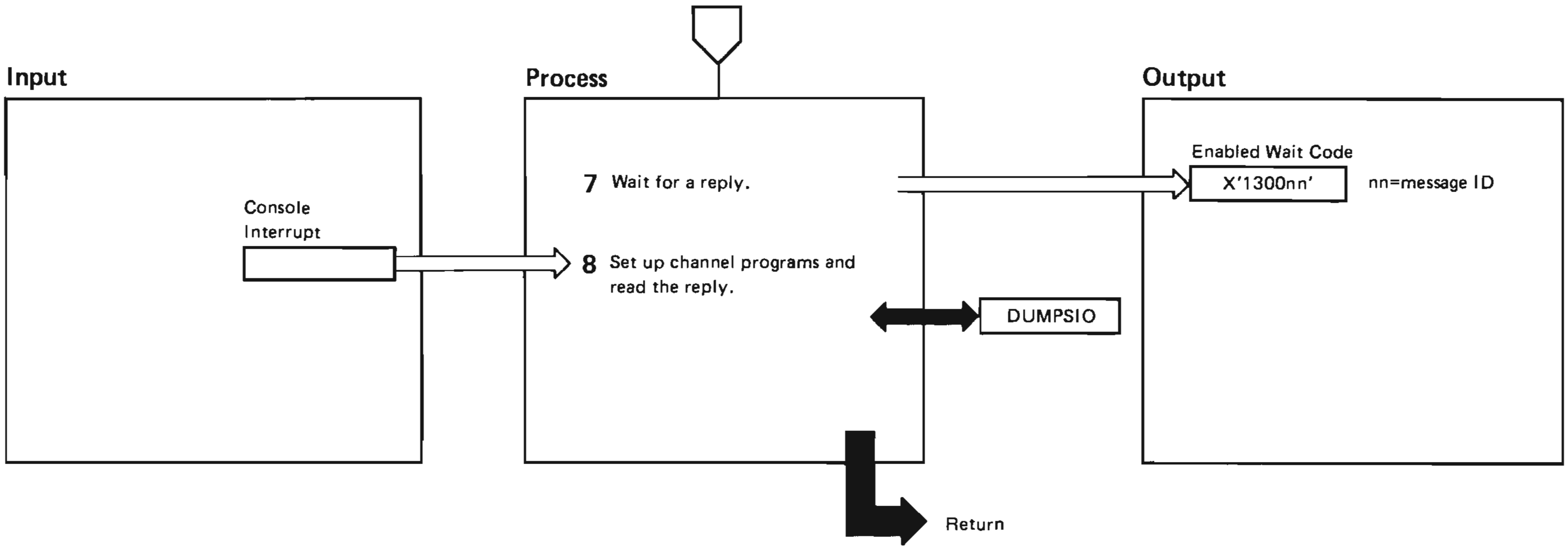
CTLINES: CTSCREND – Erase Screen Automatically

Message to Console

Diagram SADMP-27. CONSOLE – Subroutine of AMDSARDM – Read from and Write to the Operator Console (Part 2 of 4)

Extended Description	Module	Label
1 If no console is available, CONSOLE returns to its caller with a return code of 4.		
2 CONSOLE enables only for console interruptions by setting bits in control register 6.		
3 CONSOLE selects a message to write by examining the message parameter list (MSGPARM). The address of the message parameter list is passed to CONSOLE in register 6. To determine whether the message fits on the screen, CONSOLE computes the position of the message on the screen.		
4 If the screen does not have room for the message, CONSOLE checks whether to prompt the operator before erasing the screen. If prompting is specified, CONSOLE issues message AMD029D. Whether or not prompting is specified, CONSOLE then erases the screen.		
5 CONSOLE fills in the channel program and calculates the console buffer address from the screen line number.		
6 CONSOLE calls subroutine DUMPSIO to write the message to the console. If an error occurs, CONSOLE returns to the caller with a return code of 8.		

Diagram SADMP-27. CONSOLE – Subroutine of AMDSARDM – Read from and Write to the Operator Console (Part 3 of 4)



Extended Description	Module	Label
7 If the written message prompts for a reply, CONSOLE loads an enabled wait state to wait for a reply.		
8 When CONSOLE receives a console interrupt, CONSOLE fills in the channel program and calls subroutine DUMPSIO to read the reply. If an error occurs, CONSOLE returns to the caller with a return code of 8. If no errors occur, CONSOLE returns to the caller with a return code of 0.		

Diagram SADMP-28. DUMPSIO – Subroutine of AMDSARDM – Perform I/O (Part 1 of 4)

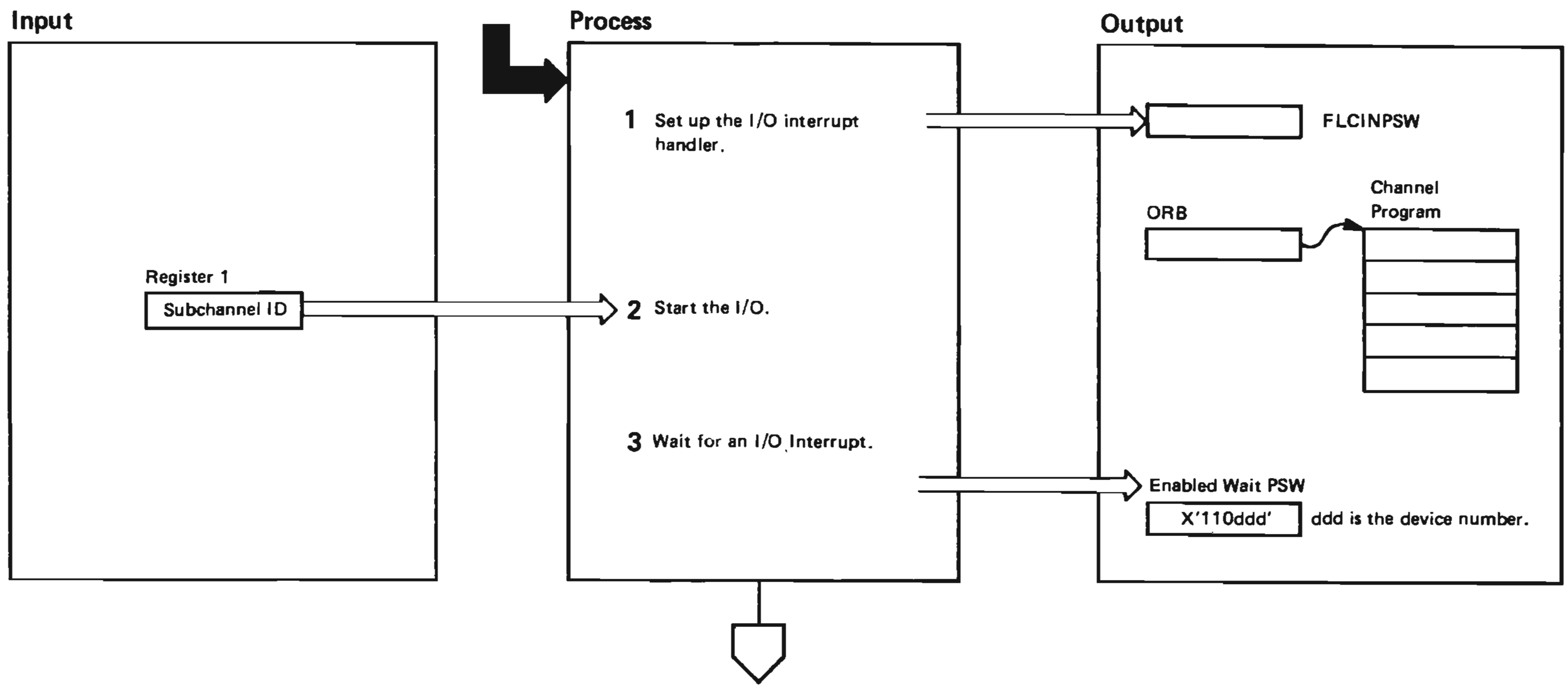


Diagram SADMP-28. DUMPSIO – Subroutine of AMDSARDM – Perform I/O (Part 2 of 4)

Extended Description	Module	Label
1 DUMPSIO sets up the I/O new PSW so that DUMPSIO can intercept I/O interrupts.		
2 DUMPSIO issues an SSCH instruction to start the I/O. If the device is not operational, DUMPSIO processes a restart to start the dump again.		
3 DUMPSIO loads a wait state PSW to wait for the I/O to complete.		

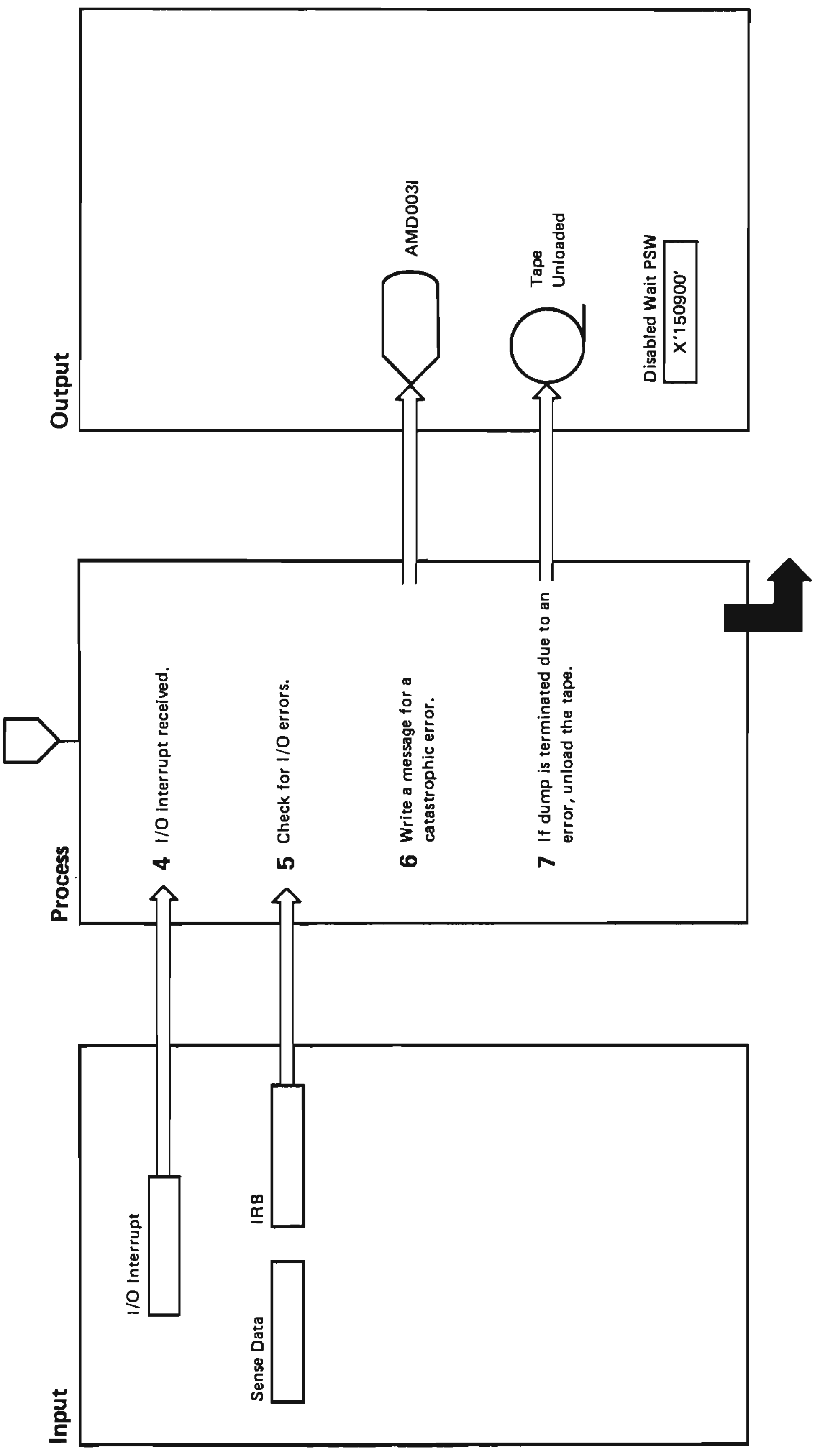

Input
I/O Interrupt
IRB
Sense Data
Process
4 I/O Interrupt received.
5 Check for I/O errors.
6 Write a message for a catastrophic error.
7 If dump is terminated due to an error, unload the tape.
Output
AMD003I
Tape Unloaded
Disabled Wait PSW
X'150900'

Diagram SADMP-28. DUMPSIO — Subroutine of AMDSARDM — Perform I/O (Part 4 of 4)

Extended Description	Module	Label
4 When DUMPSIO receives the I/O interrupt, DUMPSIO checks to see if the I/O is complete. If the I/O is not complete, DUMPSIO takes appropriate action by either restarting the I/O or waiting for further status.		
5 DUMPSIO checks for I/O errors and performs appropriate error recovery activities.		
6 If an uncorrectable error occurs, DUMPSIO issues message AMD003I.		
7 If the dump terminates because of the I/O error, and the I/O error was not on the output tape, DUMPSIO writes an end-of-file indicator on the tape, and rewinds and unloads the tape. DUMPSIO loads a wait state PSW to notify the operator that the dump ended abnormally.		

Diagram SADMP-29. LBL CHECK – Subroutine of AMDSARDM (Part 1 of 2)

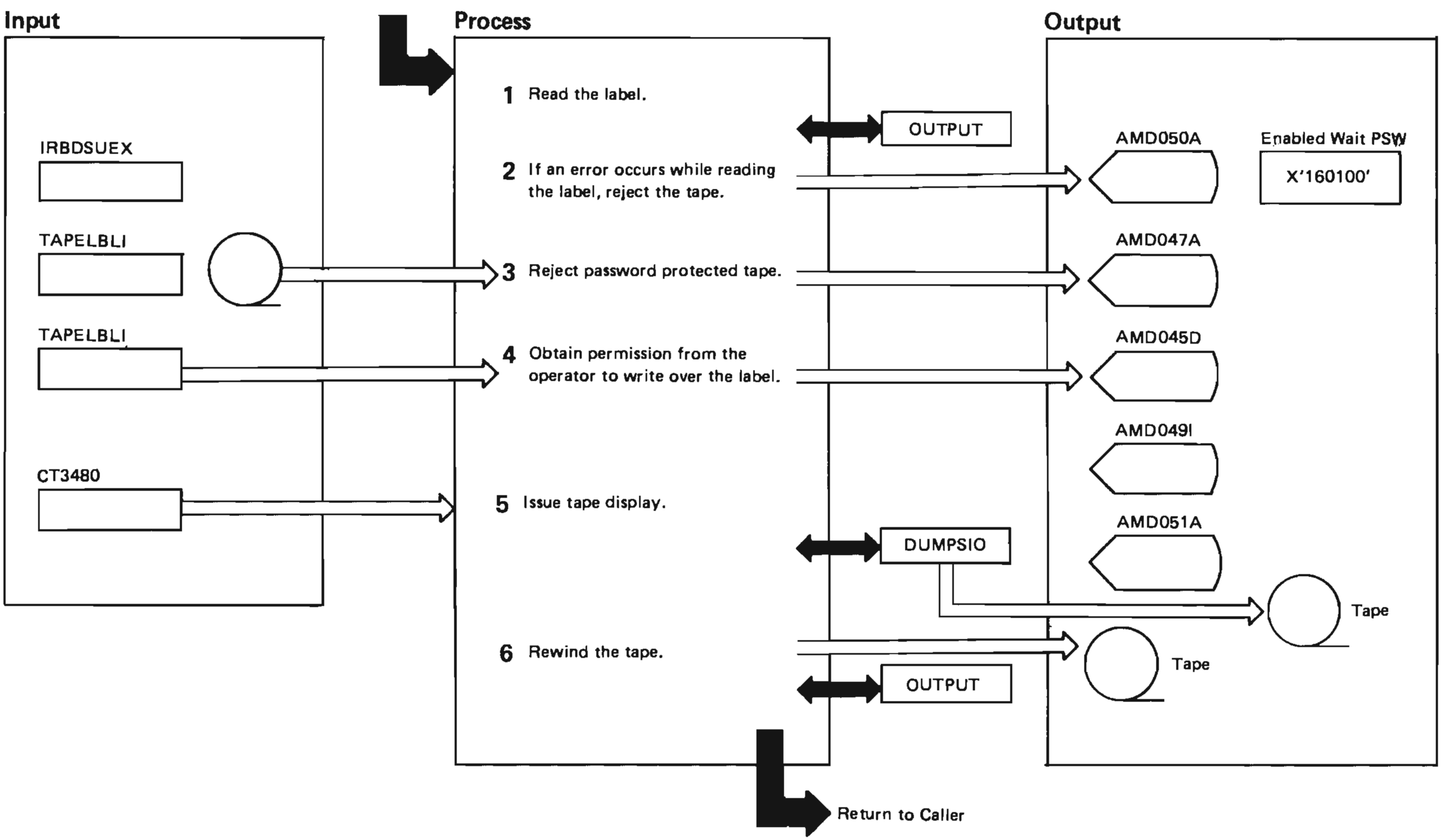


Diagram SADMP-29. LBL CHECK – Subroutine of AMDSARDM (Part 2 of 2)

Extended Description	Module	Label
1 LBLCHECK calls the OUTPUT subroutine to read the tape label.		
2 If the tape has a label that cannot be read, LBLCHECK rejects the tape and issues message AMD050A to request mounting of another tape.		
3 If the tape is password protected, LBLCHECK rejects the tape and issues message AMD047A to request mounting of another tape.		
4 If the tape has a readable label, and is not password protected, LBLCHECK issues message AMD045D to ask the operator whether or not to use the labeled, unprotected tape. If the operator provides an incorrect reply to message AMD045D, LBLCHECK issues message AMD049I to inform the operator of the syntax error. If the operator rejects the tape, LBLCHECK issues message AMD051A to ask the operator to mount another tape, and repeats steps 1-4.		
5 If any of the above tests fail or the tape is rejected, and the output is to a 3480 tape drive, LBLCHECK writes a display message to the tape drive indicating that the labelled tape is rejected and that a new tape should be mounted.		
6 If the tape is unlabeled, or if the tape passes the above tests, LBLCHECK calls the OUTPUT subroutine to determine whether the tape is file protected. If the tape is file protected, OUTPUT prompts the operator to put a ring in the tape. OUTPUT also rewinds the tape, so that the output tape is ready to use.		

Diagram SADMP-30. OUTPUT – Subroutine of AMDSARDM (Part 1 of 4)

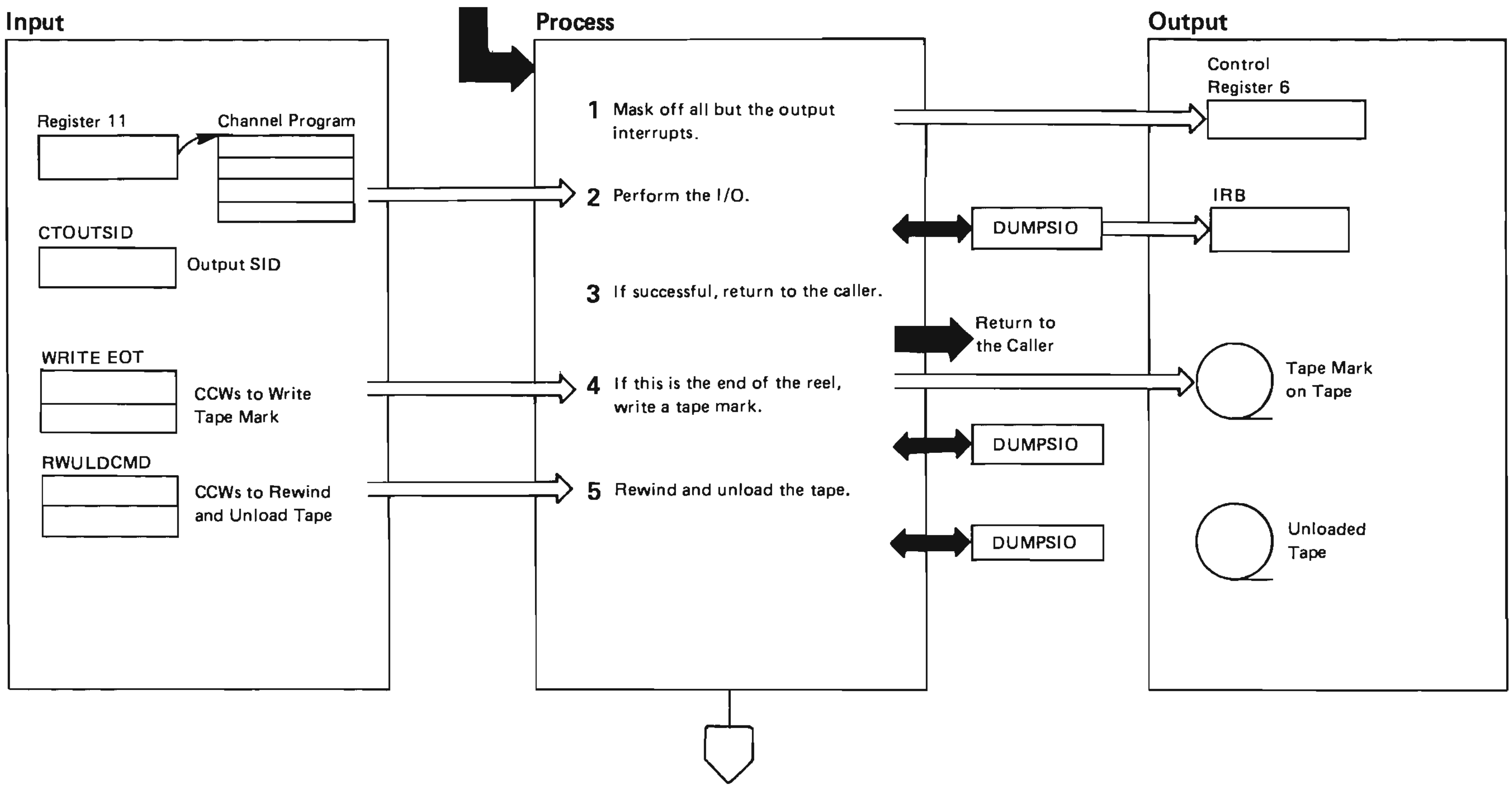


Diagram SADMP-30. OUTPUT – Subroutine of AMDSARDM (Part 2 of 4)

Extended Description	Module	Label
1 OUTPUT enables for output tape interrupts by setting bits in control register 6.		
2 OUTPUT puts the output subchannel identifier (ID) into register 1, and calls subroutine DUMPSIO to perform the I/O.		
3 If the I/O succeeds, OUTPUT returns to the caller.		
4 If OUTPUT reaches the end of the reel, OUTPUT calls subroutine DUMPSIO to write a tape mark on the tape.		
5 OUTPUT calls subroutine DUMPSIO to rewind and unload the output tape.		

Diagram SADMP-30. OUTPUT — Subroutine of AMDSARDM (Part 3 of 4)

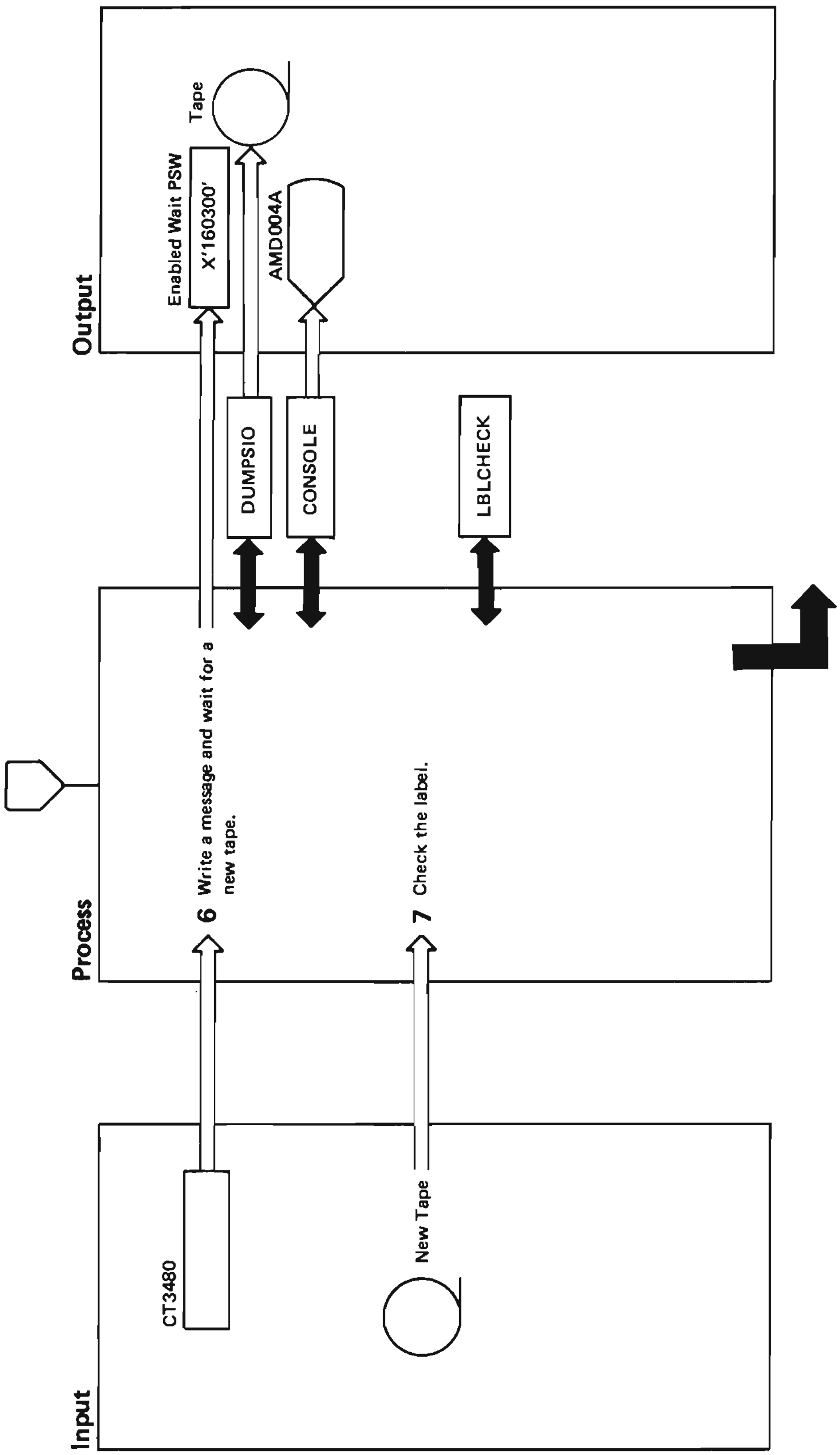


Diagram SADMP-30. OUTPUT – Subroutine of AMDSARDM (Part 4 of 4)

Extended Description	Module	Label
6 If the output is to a 3480 tape drive, output writes a display message to the output-drive indicating that the tape has reached end-of-reel, and that a new tape must be mounted.		
OUTPUT issues message AMD004A to inform the operator to mount another tape. Then OUTPUT loads a wait state PSW to wait for the new tape.		
7 After the new tape is mounted, OUTPUT calls subroutine LBLCHECK to check if the mounted tape is usable.		

Diagram SADMP-31. AMDSARSM – Real Storage Management (Part 1 of 4)

Input	Process	Output
RSMRCB, RCB Queue	1 Initialize fields and enqueue RSM RCB.	PTAVAIL: 0; RCODE: 12; RSMRCB, New RCB
RADRECNO, MAXRECNO	2 Increase and test the recursion counter.	RADRECNO: RADRECNO+1
	• If there are too many recursions, set return code and exit. ?ERROR REASON (ECRSMREC) — Error Return	Register 15: 16
	• Otherwise, accept input and set a pointer to the BCT.	Register 1 → BCT
RITFCSA, RITFPRVX, VAFAULT	3 Check the faulting address.	
VAFAULT	4 Compute the origin of the page tables and the virtual address of the page.	ORGPGTS; PGEINX; SEGINX; PXP
ORGPGTS	5 If the fault is on a page table, put the real SGTE address into the BCT.	BCTPGT: 1; MAPPEDS1

Diagram SADMP-31. AMDSARSM – Real Storage Management (Part 2 of 4)

Extended Description	Module	Label
1 AMDSARSM initializes PTAVAIL to OFF, presets an error return code, and enqueues an RSM RCB.		
2 AMDSARSM increases the recursion counter by 1 and compares this count to the maximum allowable number of recursions. If RADRECNO exceeds MAXRECNO, AMDSARSM sets an error return code of 16 and terminates. If RADRECNO does not exceed MAXRECNO, AMDSARSM accepts the input and sets a pointer to the input buffer control table.		
3 AMDSARSM checks if the faulting address is in COMMON. If it is, AMDSARSM uses the COMMON ASID for reclaiming and dumping.		
4 AMDSARSM computes the apparent origin of the faulting page. It also computes the segment index, page index, and address of the page table page.		
5 If the fault is on a page table, AMDSARSM puts the real address of the corresponding SGTE into the BCT. If the PGT itself maps a PGT, AMDSARSM marks the BCT for last-resort stealing only.		

Diagram SADMP-31. AMDSARSM – Real Storage Management (Part 3 of 4)

Input	Process	Output
PGTINV	6 Verify the DAT tables.	Register 15: 12
	• If the DAT is invalid, set a return code and exit.	Return
	7 Place the address of the page's PGTE in the BCT.	BCT: BCTPGTEP
PGTE	8 Place the address of any BCT for the page's page table in the BCT.	BCT: BCTPTBCT
	9 Locate the page in storage.	
	a. Try to reclaim the page from real storage and set a return code. (RECLAIM)	Register 15: 4 — If Reclaim
	b. Search for the page in the MPE's for this address space.	Register 15: 8 — Valid MPE found; Register 15: 12 — Invalid MPE found
	c. Check for page on the swap data set and set a return code. (RECLAIM)	Register 15: 4 — If Reclaim OR Register 15: 8 — If No Reclaim
	d. Search for the page on extended storage and page it in. (SVC (SVCXSM))	
	e. Check for page on page data set.	Register 15: 0; Register 15: 8
	– If page is on a page data set.	
	– If page is not on a page data set.	Register 15: 12
RSMRCB, RCB Queue	10 Dequeue the RSM RCB, decrease recursion count and return.	Original RCB
RADRECNO		RADRECNO: RADRECNO-1

Diagram SADMP-31. AMDSARSM — Real Storage Management (Part 4 of 4)

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

6 AMDSARSM verifies that it is using the correct DAT tables. If the DAT tables are invalid, AMDSARSM sets an error return code of 12 and terminates.

7 AMDSARSM puts the virtual address of this frame's PGTE into the BCT.

8 If the page's page table is in a buffer that is mapped by a SADMP BCT, AMDSARSM puts the address of that BCT into the page's BCT.

9 AMDSARSM locates the page in real storage, extended storage or auxiliary storage.

a. AMDSARSM tries to reclaim the page from real storage. If the page is reclaimed, AMDSARSM sets a return code of 4.

b. AMDSARSM searches the MPEs for this address space for an MPE corresponding to this page. If it finds one and the MPE is valid, AMDSARSM indicates the page can be paged in from auxiliary storage and sets a return code of 8. If it finds one and the MPE is invalid, the page cannot be found; AMDSARSM then sets a return code of 12.

c. If AMDSARSM could not reclaim the page from real storage, it looks for the page on a swap data set or in real storage where the page has just been read in by an MVS/XA swap in. AMDSARSM sets a return code of 4 if this reclaim succeeds or a return code of 8 if reclaim fails.

d. The page may also be on extended storage; if it is, AMDSARSM calls AMDSAXSM to page in the page and sets a return code 0.

If the page is on extended storage, AMDSAXSM pages it in.

e. If the page is still not found, AMDSARSM looks for the page on a page data set. If the page is on a page data set, AMDSARSM sets a return code of 8. If the page is not on a page data set, AMDSARSM sets a return code of 12.

10 AMDSARSM dequeues the RSM RCB, decreases the recursion counter, and returns to the caller.

Diagram SADMP-33. AMDSASIN – Address Space Initialization (Part 1 of 4)

Input

Register 1

ASCB

SINRCB

ASCBASID

MSTRASID

RABSWOUT

RABSWPR

SGTE

Old Top Page Table

Old 0 Page Table

RADRECNO

Process

1 Receive the input ASCB, enqueue the SINRCB, and check the validity of the RAB.

?VCHK

2 If the address space is the master scheduler address space, set a return code.

3 If the address space is not the master scheduler address space:

- Determine the location of the address space.
- Create a segment table.
- Create the top page table.
- Create the 0 page table.

4 Initialize the RSM function and the address fault recursion count.

Output

SINRCB

New RCB

Register 15

0

SGTE

Top Page Table

0 Page Table

RADRECNO

0

Extended Description	Module	Label

AMDSASIN initializes an address space so that SADMP can execute in that address space.

1 AMDSASIN receives the input ASCB, enqueues a SIN RCB, and checks the validity of the RAB.

2 If the address space is the master scheduler address space, AMDSASIN sets a return code of 0.

3 If the address space is not the master scheduler address space:

- AMDSASIN determines whether the address space is swapped in, swapped out, or being processed by a MVS/XA swap.
- AMDSASIN creates the segment table first by reading and dumping the segment table, and second, by copying all common SGTES from the master SGT into the new segment table.
- AMDSASIN creates the top page table, reads in and dumps the top page table, and marks the top page table entries as invalid. AMDSASIN hooks up the top PGT so that the swapped-in address space can function.
- AMDSASIN creates the 0 page table, reads in and dumps the 0 page table, and marks the 0 page table entries as invalid. AMDSASIN hooks up the page 0 DAT tables.

4 AMDSASIN initializes the RSM function for local address space by determining the beginning and ending addresses of the virtual storage ranges used by page tables in the local address space. AMDSASIN initializes the address fault recursion count.

Diagram SADMP-33. AMDSASIN — Address Space Initialization (Part 3 of 4)

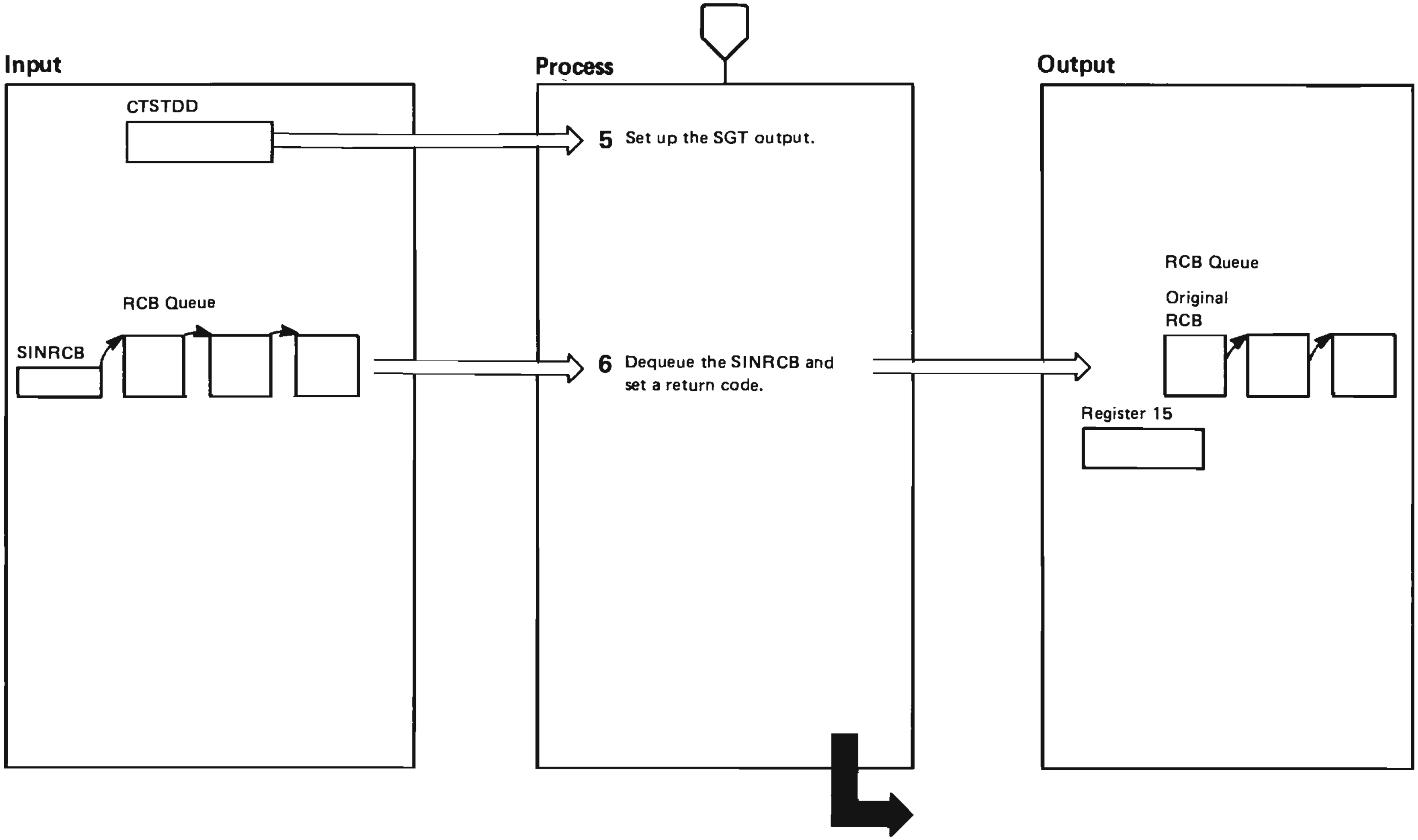


Diagram SADMP-33. AMDSASIN – Address Space Initialization (Part 4 of 4)

Extended Description	Module	Label
5 AMDSASIN sets up the SGT output by determining the value to place in CTSTDD, the portion of the CCT that is the STD for the address space being dumped.		
6 AMDSASIN dequeues the SIN RCB and sets the return code. A return code of 0 indicates that the address space was already in, and no swapping is necessary. A return code of 4 indicates that the address space was swapped in. A return code of 8 indicates abnormal termination; the address space was not swapped in.		

Diagram SADMP-34. AMDSASIO – Utility for I/O Operations (Part 1 of 2)

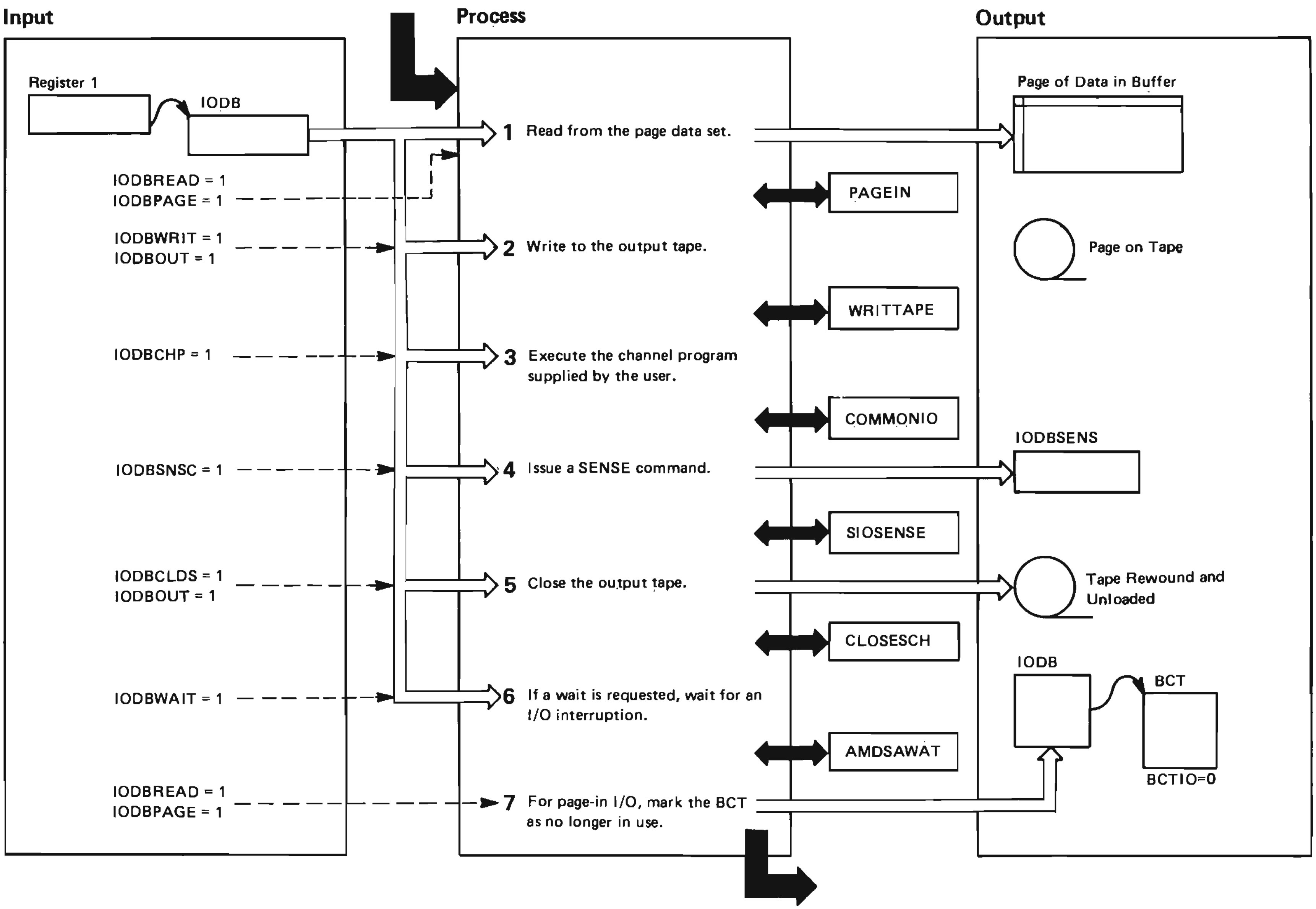


Diagram SADMP-34. AMDSASIO — Utility for I/O Operations (Part 2 of 2)

Extended Description	Module	Label
1 If a caller requests that a record be read from auxiliary storage (IODBREAD = 1 and IODBPAGE = 1), AMDSASIO calls the PAGEIN subroutine.		
2 If a caller requests that a record be written to the output tape (IODBWRIT = 1 and IODBOUT = 1), AMDSASIO calls the WRITTAPE subroutine.		
3 If the caller requests that I/O be executed using the caller's supplied channel program (IODBCHP), AMDSASIO calls the COMMONIO subroutine.		
4 If the caller requests that a SENSE command be issued (IODBSNSC = 1), AMDSASIO calls the SIOSENSE subroutine.		
5 If the caller requests that a CLOSE be issued for the output tape (IODBCLOS = 1 and IODBOUT = 1), AMDSASIO calls the CLOSESCH subroutine.		
6 If the caller requests AMDSASIO to wait for an I/O interrupt (IODBWAIT = 1), AMDSASIO calls the AMDSAWAT subroutine. The request is valid only if the return code value is zero. That is, when starting I/O, no error occurred.		
7 If the request was for a read from auxiliary storage (IODBREAD = 1 and IODBPAGE = 1), AMDSASIO frees the buffer control table (BCT) entry by marking it as no longer in use (BCTIO = 0).		

Diagram SADMP-35. PAGEIN – Subroutine of AMDSASIO (Part 1 of 4)

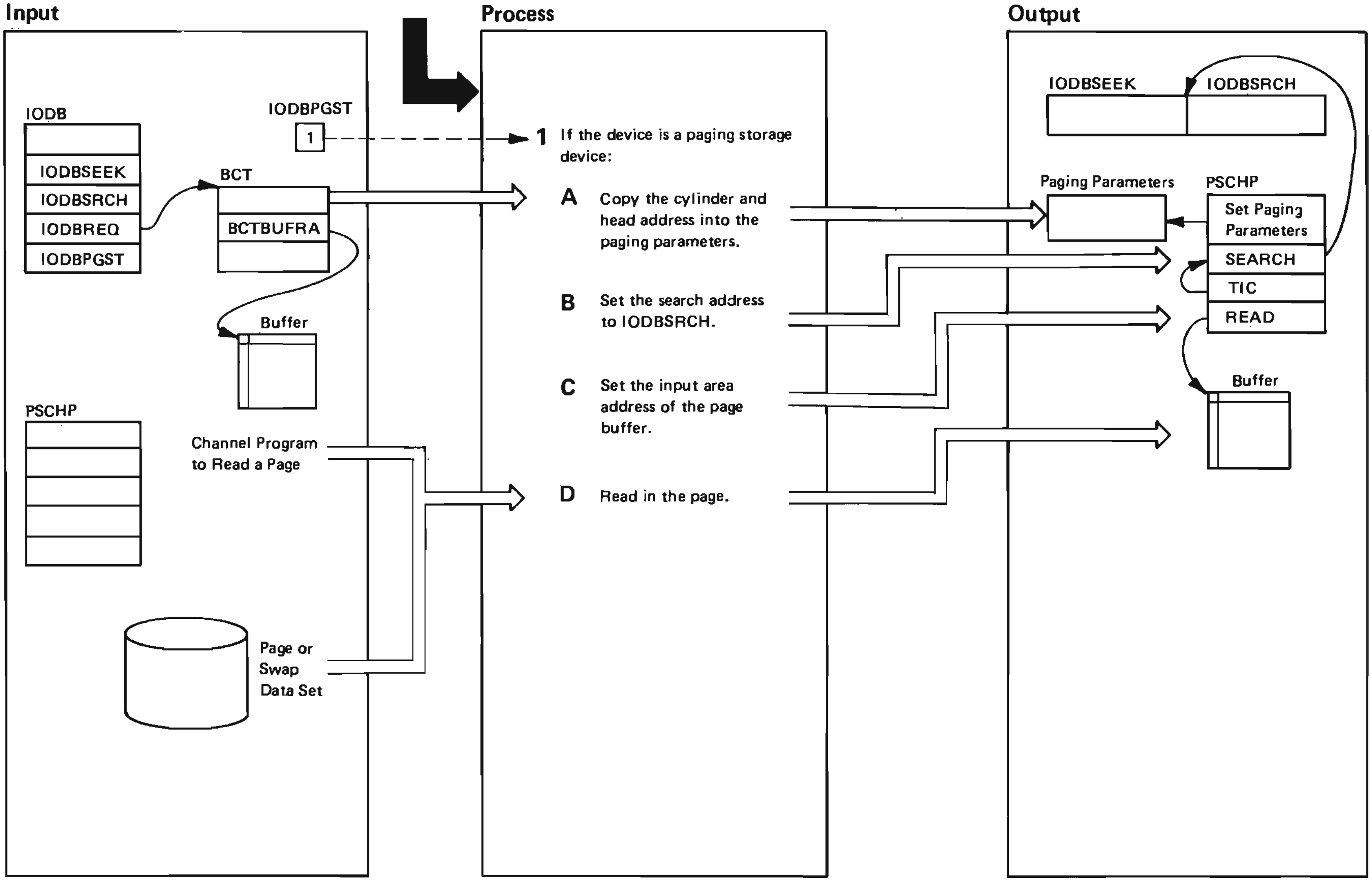


Diagram SADMP-35. PAGEIN – Subroutine of AMDSASIO (Part 2 of 4)

Extended Description	Module	Label
1 When the device is a paging storage device:	AMDSASIO	PAGEIN
A PAGEIN points the cylinder and head address to the paging parameters, PSPCCHH.		
B PAGEIN puts the pointer to the search address into the channel program search address.		
C PAGEIN puts the real address of the page buffer into the channel program input area address.		
D PAGEIN calls the COMMONIO subroutine passing it a channel program (PSCHP). COMMONIO executes the channel program causing a page to be read in from auxiliary storage.		

Diagram SADMP-35. PAGEIN – Subroutine of AMDSASIO (Part 3 of 4)

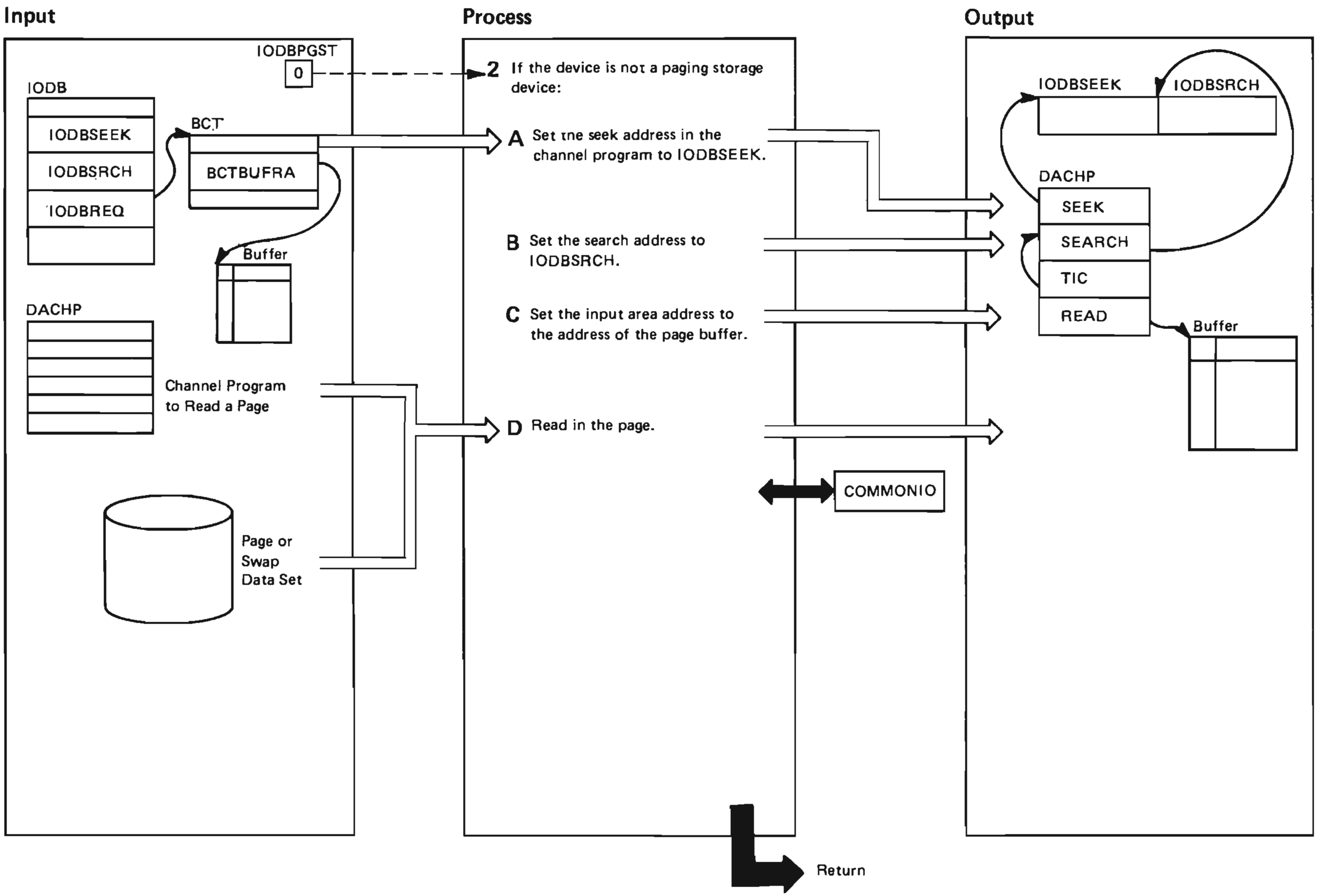


Diagram SADMP-35. PAGEIN – Subroutine of AMDSASIO (Part 4 of 4)

Extended Description	Module	Label
2 When the device is not a paging storage device:	AMDSASIO	PAGEIN
A PAGEIN puts the pointer to the SEEK address into the channel program SEEK address.		
B PAGEIN places the pointer to the search address into the channel program search address.		
C PAGEIN places the real address of the page buffer into the channel program input area address.		
D PAGEIN calls the COMMONIO subroutine passing it a channel program (DACHP). COMMONIO executes the channel program causing a page to be read in from auxiliary storage.		

Diagram SADMP-36. CLOSESCH – Subroutine of AMDSASIO (Part 1 of 2)

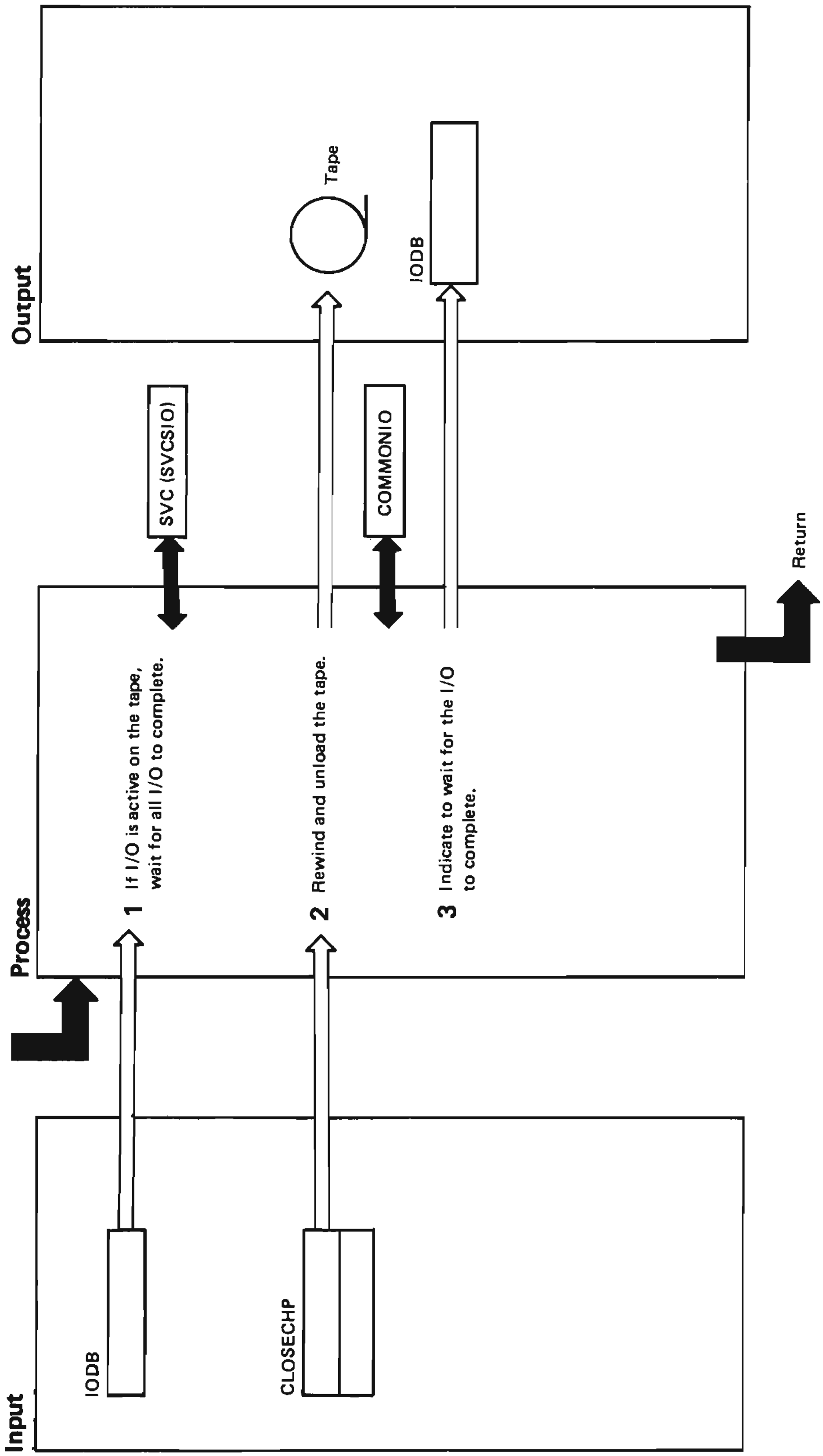


Diagram SADMP-36. CLOSESCH — Subroutine of AMDSASIO (Part 2 of 2)

Extended Description	Module	Label
1 If I/O is active on the tape (IODBINTX=1), CLOSESCH waits for all I/O to complete. This is accomplished by a recursive call to AMDSASIO via an SVC instruction. Control is returned to CLOSESCH when all I/O has completed (IODBINTX=0).	AMDSASIO	CLOSESCH
2 CLOSESCH calls the COMMONIO subroutine, passing it a channel program (CLOSECHP). COMMONIO executes the channel program causing the output tape to be rewound and unloaded.		
3 CLOSESCH sets an indicator (IODBWAIT=1) so that mainline AMDSASIO will wait for the I/O that was started in step 2 to complete.		

Diagram SADMP-37. SIOSENSE – Subroutine of AMDSASIO (Part 1 of 2)

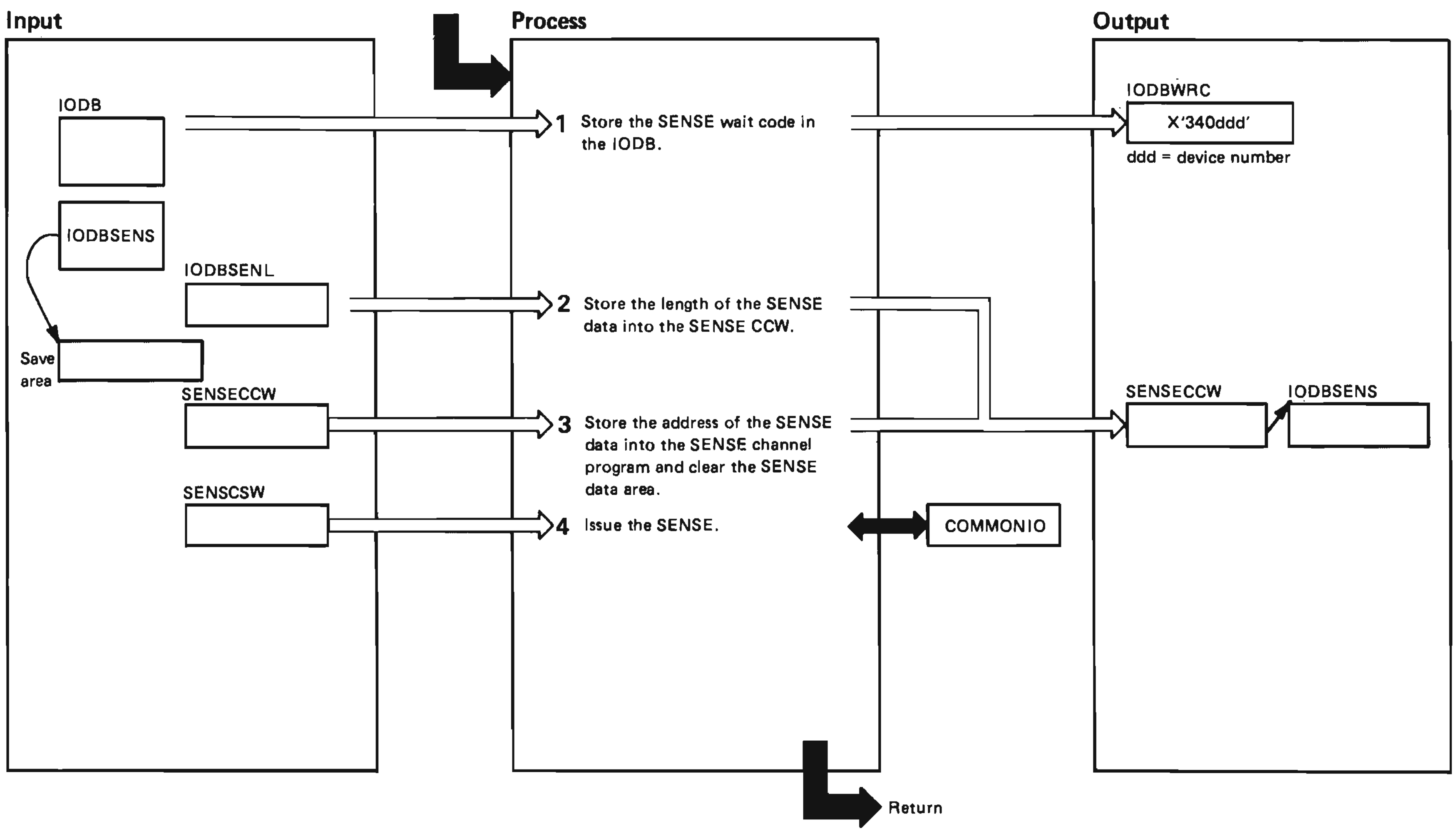


Diagram SADMP-37. SIOSENSE – Subroutine of AMDSASIO (Part 2 of 2)

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

1 SIOSENSE places the SENSE wait state code and the device number into the IODB.

2 SIOSENSE copies the length IODBSENL of the sense data into the length field of the sense CCW.

3 SIOSENSE places a pointer to the SENSE information area into the SENSE channel program and clears the SENSE information area.

4 SIOSENSE calls the COMMONIO subroutine, passing it a channel program (SENSECCW). COMMONIO executes the channel program causing a SENSE command to be issued for the device indicated by IODBDEV.

Diagram SADMP-38. COMMONIO – Subroutine of AMDSASIO (Part 1 of 2)

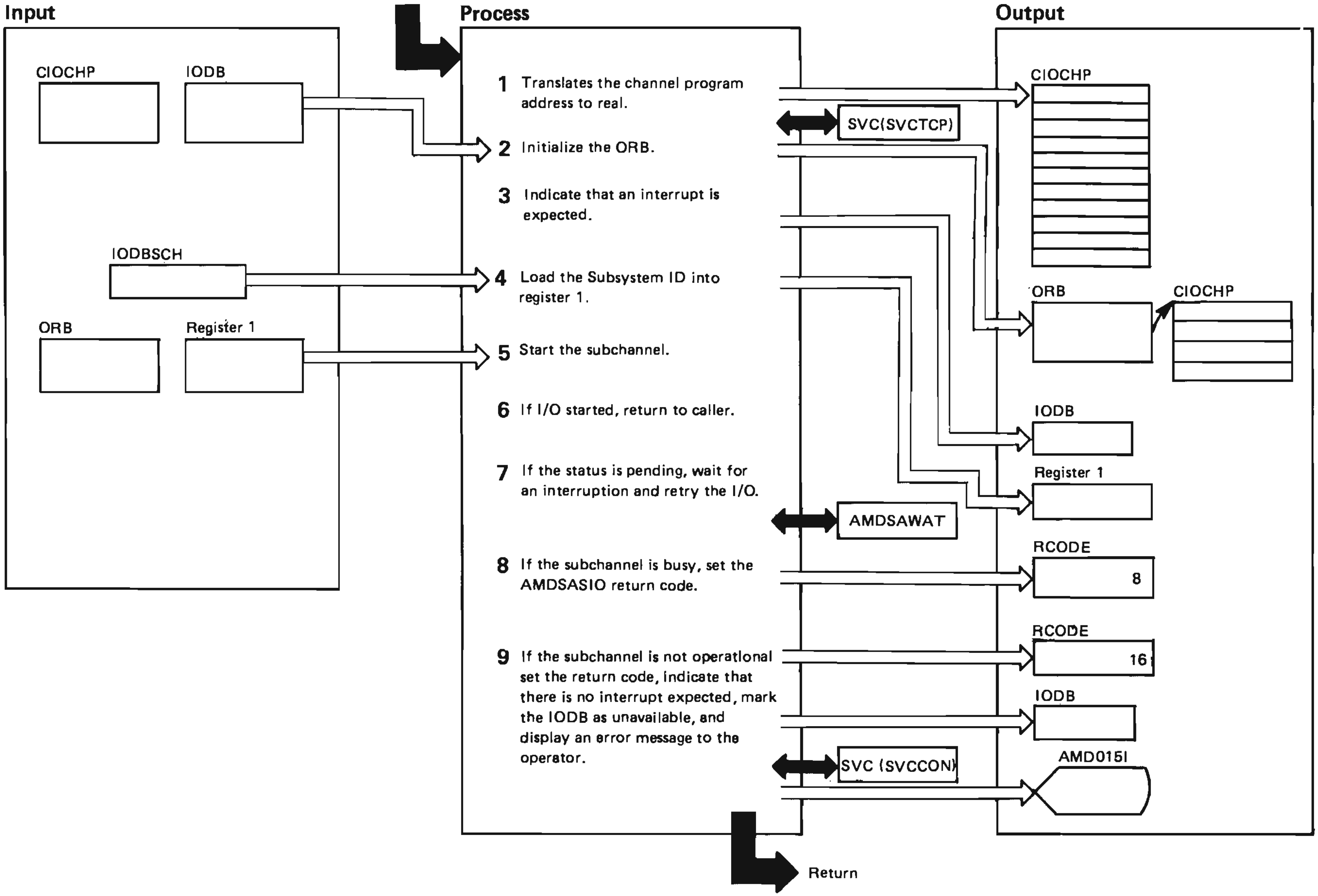


Diagram SADMP-38. COMMONIO – Subroutine of AMDSASIO (Part 2 of 2)

Extended Description	Module	Label

1 If the channel program has virtual addresses (IODBCHPR = 0), COMMONIO calls AMDSATCP to translate the addresses to real.

2 COMMONIO initializes the operation request block (ORB) by setting a pointer to the IODB and the channel program (CIOCHP). COMMONIO indicates that the CCWs are format 1 CCWS, and that all paths should be enabled for the device.

3 COMMONIO indicates that an I/O interrupt is expected (IODBINTX=1).

4 COMMONIO places the subsystem identifier ('SID') into register 1, where it is used by the start sub-channel command (SSCH).

5 COMMONIO issues the SSCH command passing it the ORB.

6 If the SSCH command was successful, COMMONIO had no more processing to do. It returns to the caller.

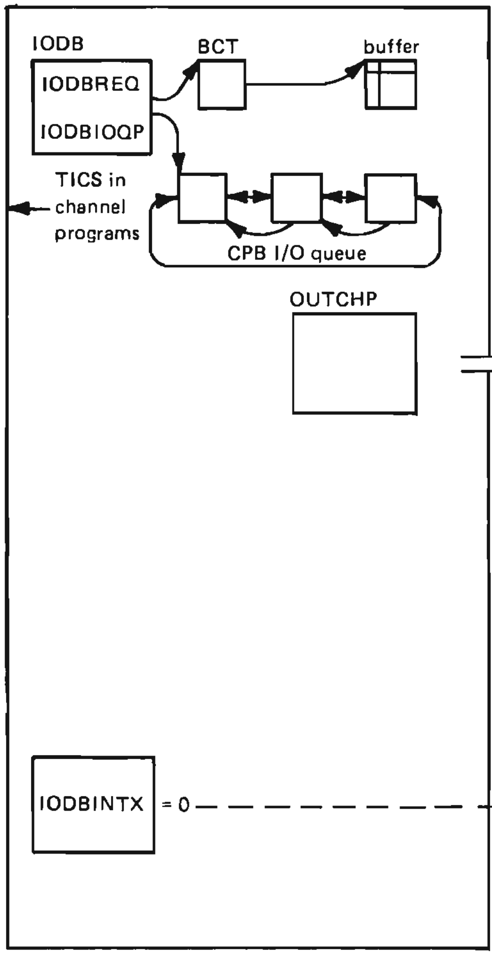
7 If the SSCH command indicates that the subchannel has status pending, COMMONIO calls the subroutine AMDSAWAT to handle the I/O interrupt. When AMDSAWAT completes, COMMONIO executes from step 2 to retry the SSCH command.

8 If the SSCH command indicates that the subchannel is busy, (I/O has already been started to that sub-channel), COMMONIO sets the return code to 8.

9 If the SSCH command indicates that the subchannel is not operational, COMMONIO sets the return code to 16. COMMONIO indicates that no I/O interrupts are expected (IODBINTX=0) and that the IODB is unavailable (IODBUNAV=1). COMMONIO issues message AMD015I and if the error occurred on the output volume, it issues message AMD032I. COMMONIO these messages by calling the virtual dump console service routine, AMDSACON, via an SVC instruction.

Diagram SADMP-39. WRITTAPE – Subroutine of AMDSASIO (Part 1 of 2)

Input



Process

1 Create a CPB and build the output channel program in it.

SVC (SVCGTM)

2 Hook the BCT onto the CPB, indicating that the BCT is waiting for the channel program to complete.

3 Add the CPB to the left of the output I/O queue.

4 If another CPB is on the queue, chain the current one to its right neighbor.

SVC (SVCTCP)

5 If this is the only CPB on the queue and the subchannel is inactive, start the I/O.

COMMONIO

Output

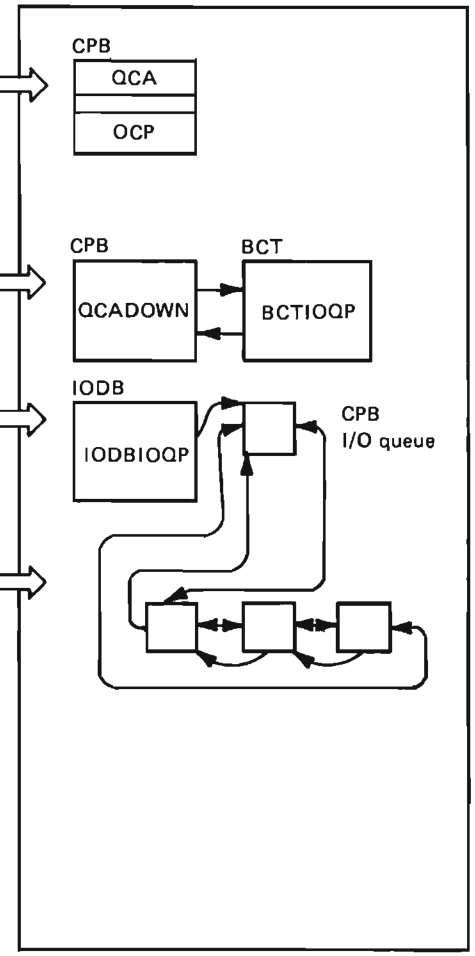


Diagram SADMP-8. WRITREAD – Subroutine of AMDSACON (Part 2 of 2)

Extended Description	Module	Label
1 WRITREAD increases the count of lines on the screen. If the screen would be filled by that amount of lines, WRITREAD sets the CCW command to ERASE/WRITE and resets CTLINES to 1.		
2 WRITREAD computes the 12/14-bit screen address from CTLINES and the known screen width (SW3277).		
3 The channel program sends a WRITE or ERASE/WRITE command to the console, unlocks the screen if a reply is expected, and furnishes the screen address and the data to be written.		
4 The ?IO macro points the console IODB (XIOBCON) to the channel program and calls AMDSASIO.		
5 If a reply is expected, WRITREAD calls AMDSASIO to wait for an attention interruption caused when the operator presses enter.		
6 WRITREAD builds a channel program to read the reply and calls AMDSASIO (Steps 3 and 4).		
7 WRITREAD converts the reply to upper case.		

Diagram SADMP-40. AMDSAWAT – Subroutine of AMDSASIO (Part 1 of 2)

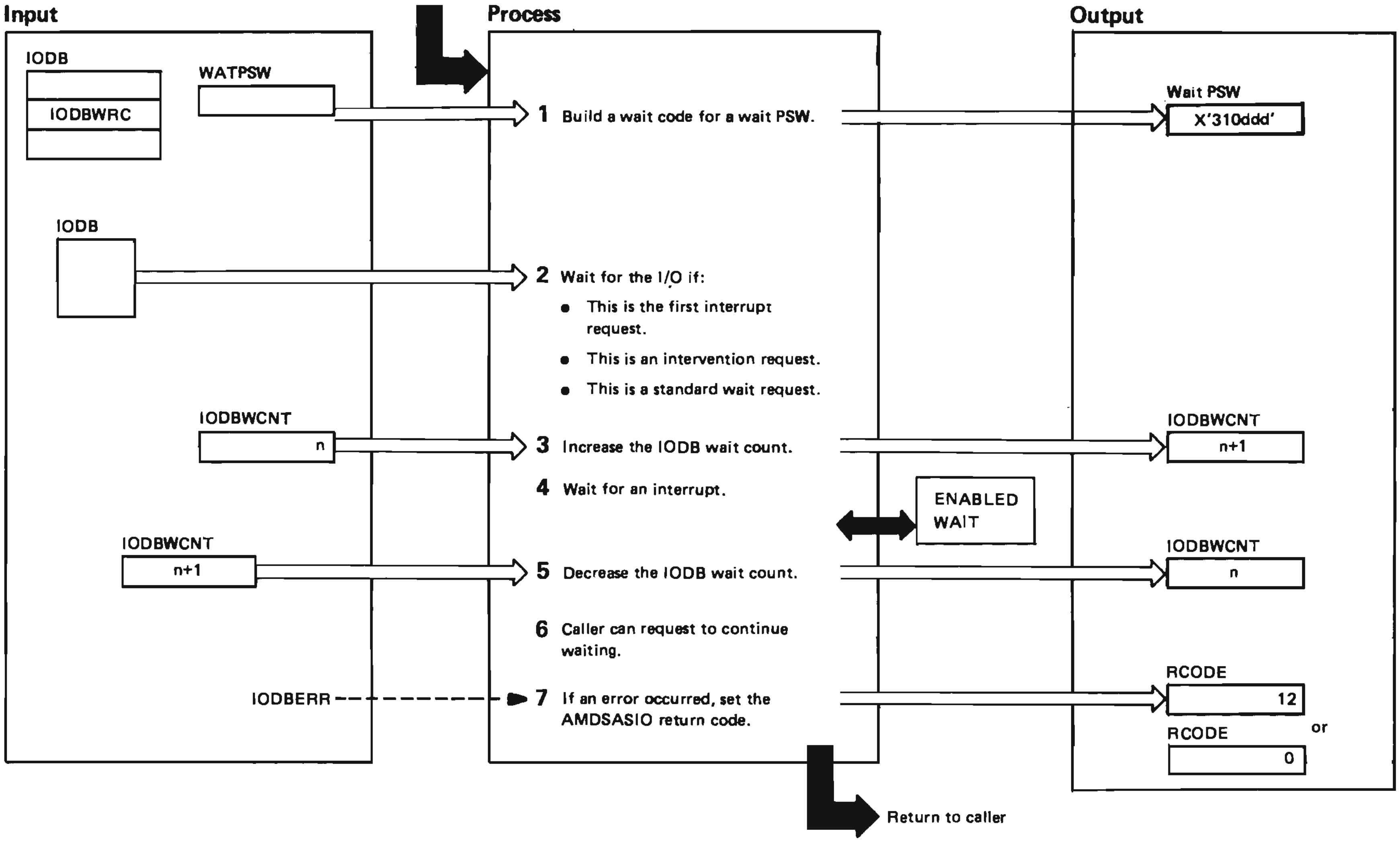


Diagram SADMP-40. AMDSAWAT — Subroutine of AMDSASIO (Part 2 of 2)

Extended Description		Module	Label
1	AMDSAWAT builds a wait state PSW. AMDSAWAT places a wait state code and the device number into the PSW. The PSW is enabled for I/O, external, and machine check interrupts.	AMDSASIO	AMDSAWAT
2	For an interrupt request (IODBINRQ = 1), AMDSAWAT waits, whether or not the device is active; it returns after the first interrupt. For a first interrupt request (IODBFINT = 1), AMDSAWAT waits for an interrupt only if the device is active, then returns once an interrupt is received. For other types of requests, AMDSAWAT waits until no more interrupts are expected (IODBINTX = 0).		
3	AMDSAWAT increases the wait count in the IODB.		
4	AMDSAWAT loads a wait state PSW (WATPSW) via the LPSW instruction.		
5	After the interrupt is received, AMDSAWAT decreases the wait count in the IODB. After the I/O has completed or there is a PCI interrupt from a tape, AMDSAIOI (AMDSADMP's virtual dump I/O interrupt handler) returns control here.	AMDSASIO	IOWATRCT
7	If the IODB shows an unrecoverable error, AMDSAWAT sets the return code to 12.		

Diagram SADMP-41. AMDSASRR–System Restart Handler (Part 1 of 4)

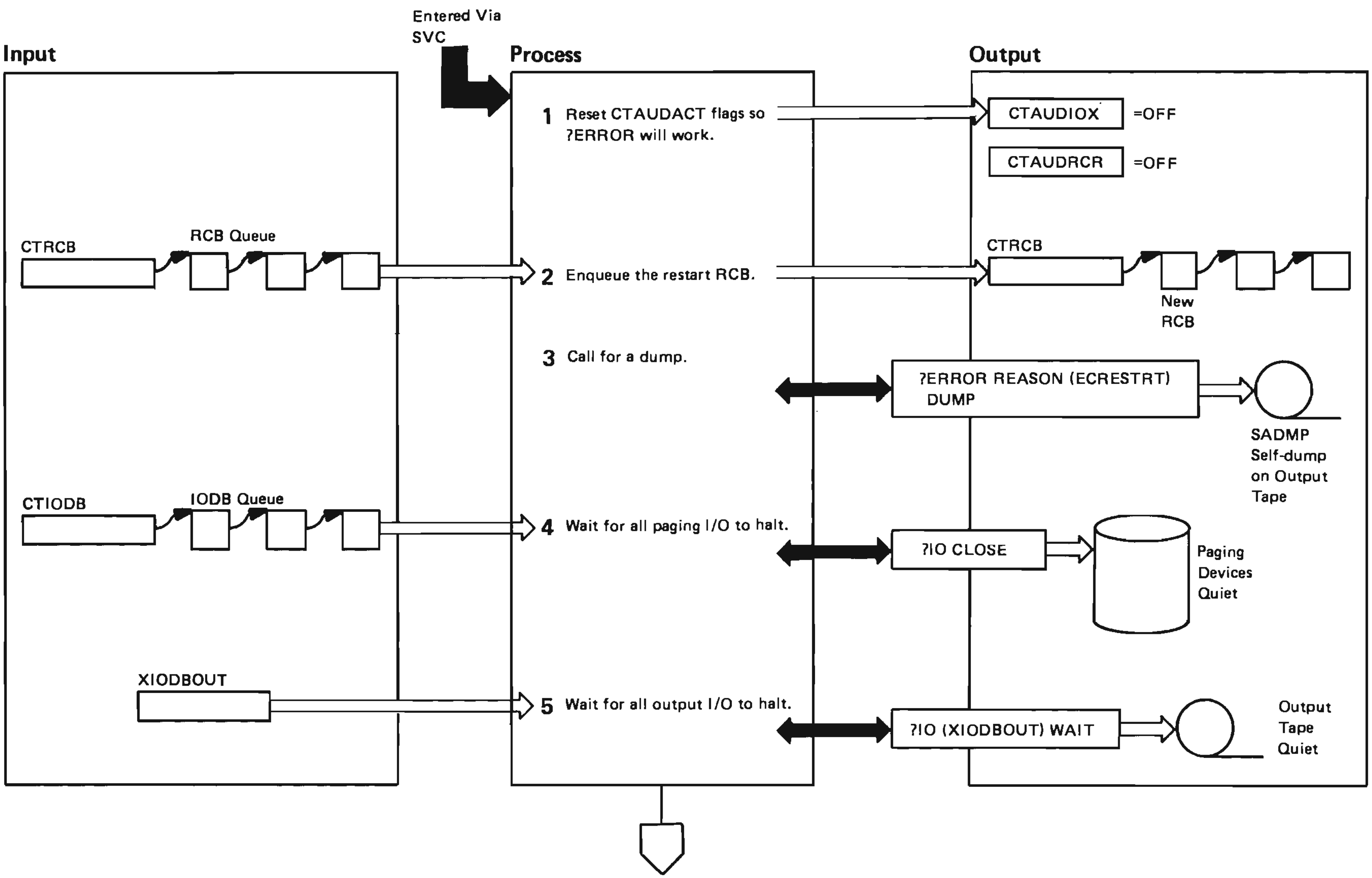


Diagram SADMP-41. AMDSASRR–System Restart Handler (Part 2 of 4)

Extended Description	Module	Label
1 AMDSASRR turns off CTAUDRCR and CTAUDIOX to show that all error recovery modules can be used.		
3 AMDSASRR sets CTAUDDMP to call for a diagnostic dump, then calls AMSSAAUD. AMDSASRR assumes the operator restarted SADMP because of an error.		
4 AMDSASRR proceeds down the IODB queue (CTIODB) looking for paging IODBs (IODBPAGE=ON). For each paging IODB that has an active device (IODBINTX=ON), AMDSASRR calls AMDSASIO to wait for the paging I/O to complete.		
5 AMDSASRR calls AMDSASIO to wait until all output I/O to the output tape has completed (IODBINTX=OFF). The output tape has an IODB of XIODBOUT.		

Diagram SADMP-41. AMDSASRR–System Restart Handler (Part 3 of 4)

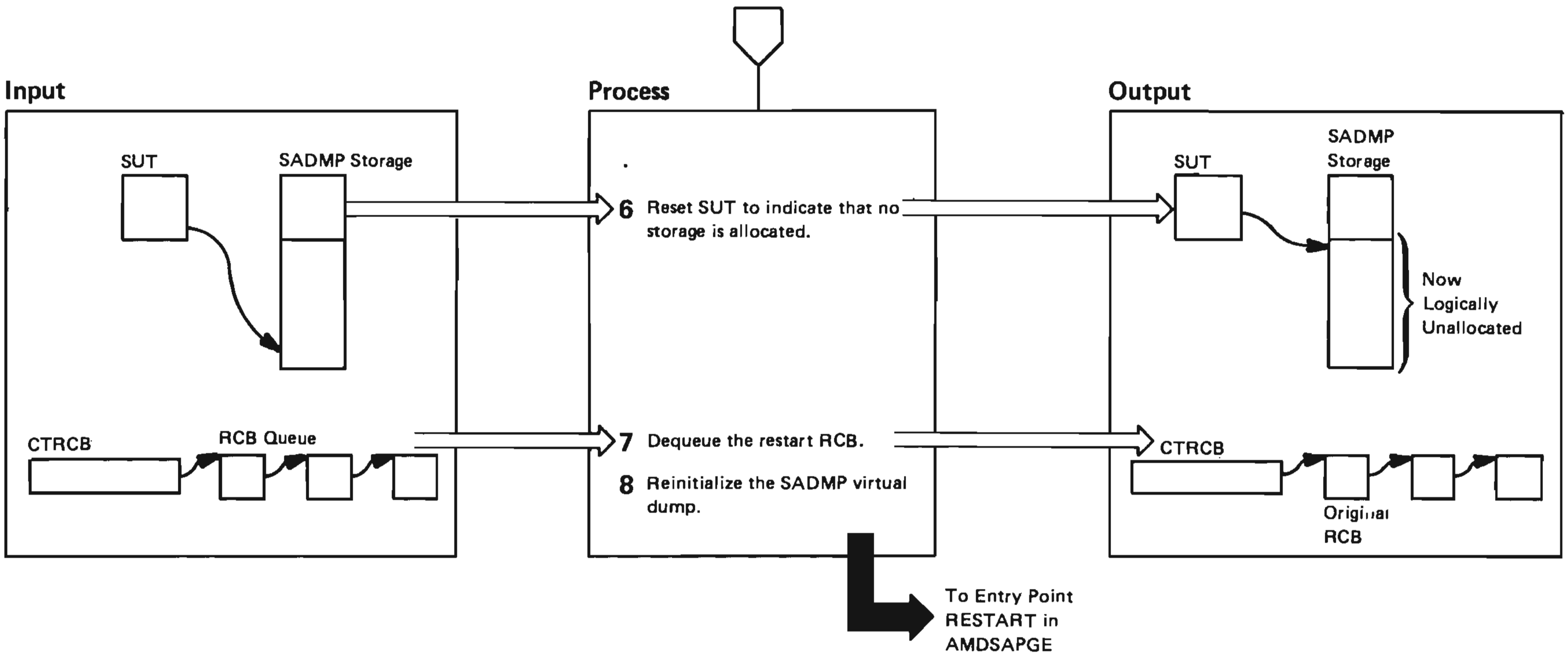
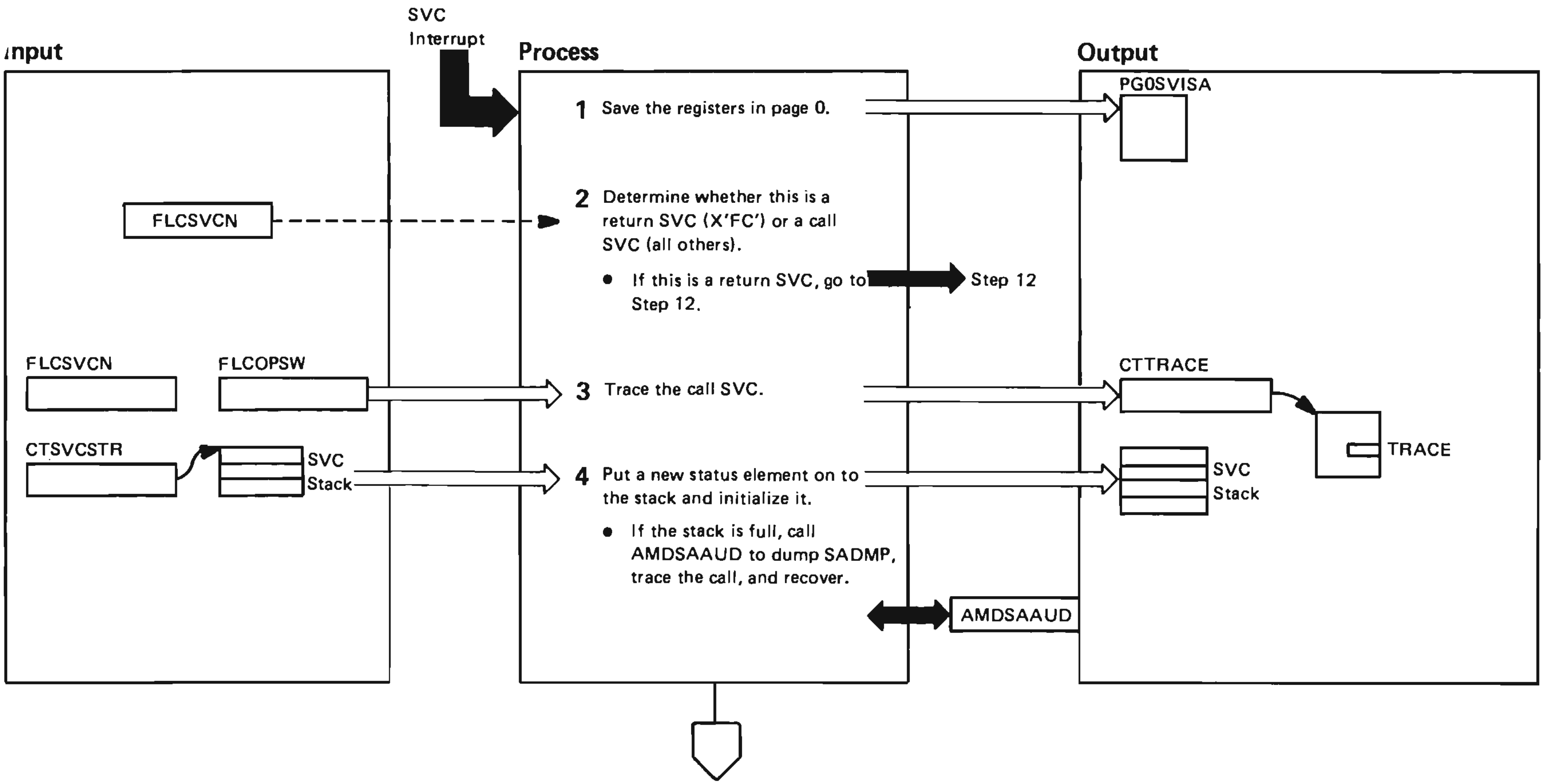


Diagram SADMP-41. AMDSASRR–System Restart Handler (Part 4 of 4)

Extended Description	Module	Label
6 AMDSASRR sets SUTFUNIN=FUNIN to indicate that no storage is allocated.		
7 AMDSASRR dequeues the restart RCB.		
8 AMDSASRR branches to label RESTART at the beginning of AMDSAPGE. This causes AMDSAPGE to reinitialize all SADMP resources and rerun the virtual dump.		

Diagram SADMP-42. AMDSASVI–SVC Interruption Handler (Part 1 of 8)



Extended Description	Module	Label
1 AMDSASVI copies the registers at entry into PG0SVISA in page 8.		
2 AMDSASVI determines whether the SVC is a return SVC (FLCSVCN = X'FC') or a call SVC (FLCSVCN ¬= X'FC'). For a return SVC, AMDSASVI goes to Step 12.		
3 AMDSASVI makes a trace table entry (CTTRACE) showing the SVC old PSW and the interrupt code FLCSVCN. CTSVCSTV points to the SVC stack (SVCSTACK), which contains SVC status entries (SVCSTAT) for the current SVC calling sequence.		
4 AMDSASVI increases SSINDEX by one to add a new stack entry. If the stack is full, AMDSASVI branch-enters AMDSAAUD to terminate the caller. AMDSASVI copies control register 1 into SSTDCR1, CTASIDA into SSASID, CTASCBA into SSASCB, the SVC old PSW (FLCSOPSW) into SSPSW, the caller's save area address (PG0SR13) into SSSAR13, the caller's return point address (PG0SR14) into SS0RIG14, and the SVC number (FLCSVCN) into SSNUM.		

Diagram SADMP-42. AMDSASVI–SVC Interruption Handler (Part 3 of 8)

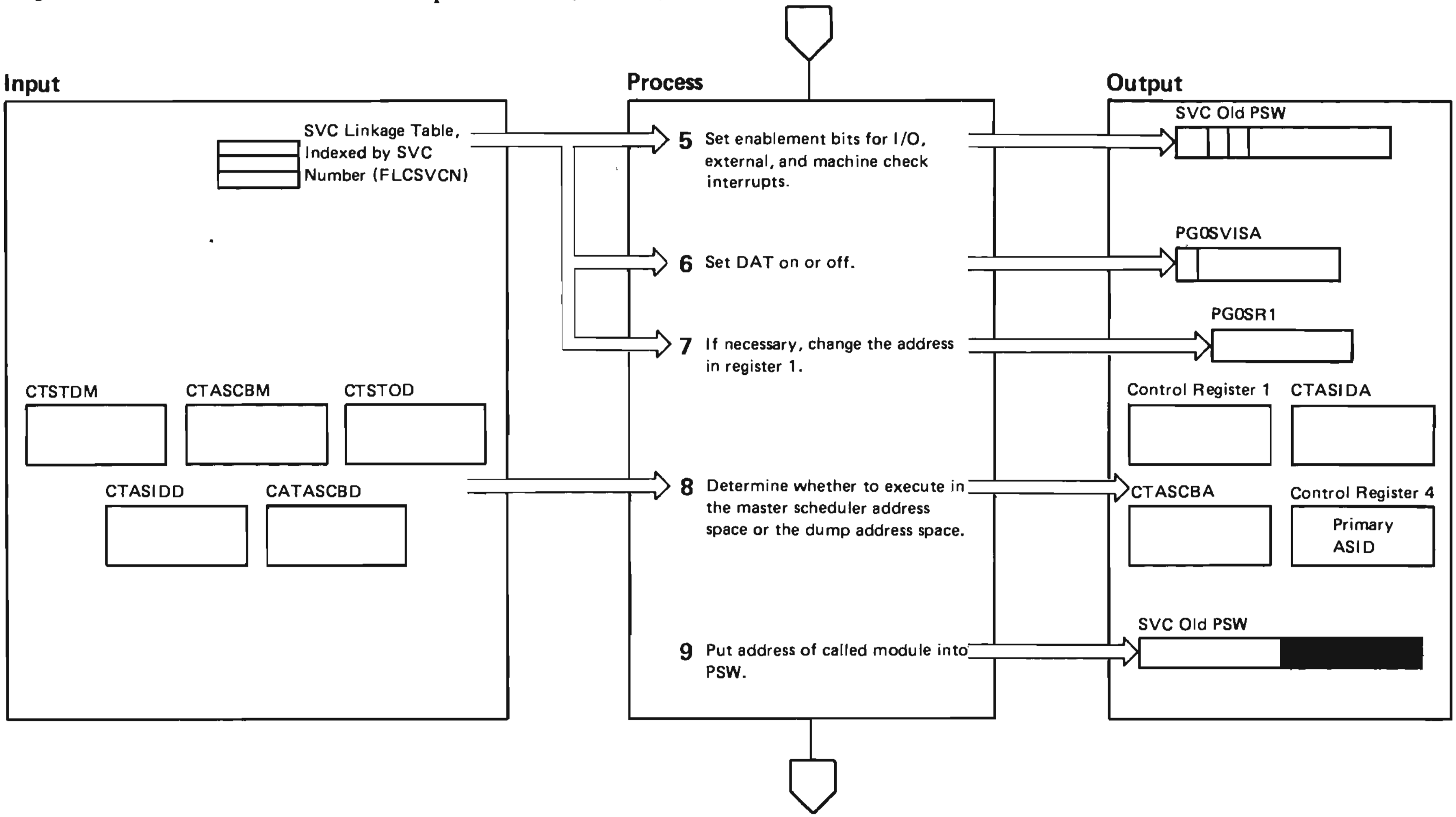
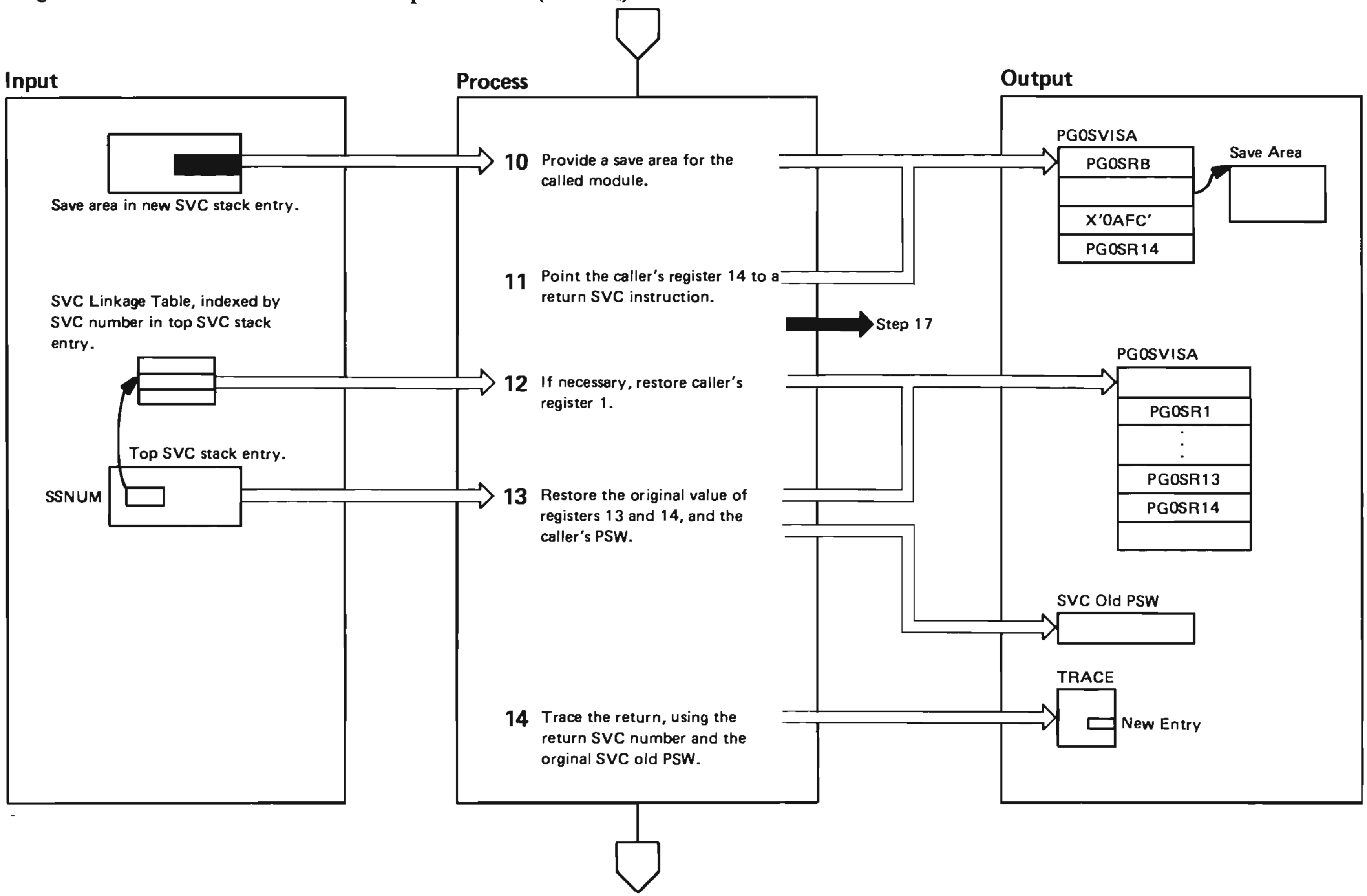


Diagram SADMP-42. AMDSASVI–SVC Interruption Handler (Part 4 of 8)

Extended Description	Module	Label
5 The called module has an entry in the SVC table SVCTABLE indexed by the SVC number. The table entry SVCLINK describes the environment and the entry point address to be given control. If SLDISABL=ON, AMDSASVI turns off the I/O, external, and machine check enablement bits in the SVC old PSW. If SLENABL=ON, AMDSASVI turns these bits on.		
6 If SLDATOFF=ON, AMDSASVI turns off the DAT bit in the SVC old PSW. If SLDATON=ON, AMDSASVI turns on the DAT bit.		
7 If SLREG1=ON, the caller is passing a parameter in register 1 that may need to be relocated because the two routines have different DAT modes. Change the address in register 1 from virtual to real or from real to virtual, if this is required.		
8 AMDSASVI determines whether to execute in the master address space or the dump address space. If the called module is DAT-on and SLASIDD=ON, the called module runs in the dumped address space. AMDSASVI sets control register 1 to CTSTDD, CTASIDA to CTASIDD and CTASCBA to CTASCBD.		

Extended Description	Module	Label
If the caller module is DAT-on, SLCR1=ON and control register 1 equals CTSTDD; the called module runs in the dumped address space.		
If the caller module is DAT-on, SLCR1=ON and control register 1 equals CTSTDM; the called module runs in the master address space. AMDSASVI sets control register 1 to CTSTDM, CTASIDA to CTASIDM and CTASDBA to CTASCBM.		
If the called module is DAT-on, SLCR1=OFF and either SLMASTER=ON or the caller is DAT-off, the called module runs in the master scheduler address space.		
If the called module is DAT-on and none of the above is true, the called module runs in the currently active address space.		
9 AMDSASVI stores SLADDR into PSWIA in the SVC old PSW.		

Diagram SADMP-42. AMDSASVI–SVC Interruption Handler (Part 5 of 8)



Extended Description	Module	Label
10 AMDSASVI points the called module's register 13 at PG0SR13 to the SVC stack entry save area SSSAVE.		
11 If SLNCR=OFF, AMDSASVI points the called module's register 14 at PG0SR14 to the return SVC instruction at PG0SVCRT register 15 to the called module's entry point.		
12 For return SVC s, AMDSASVI uses the top SVC stack entry and the SVC linkage table entry for the module being returned from.		
If SLREG1=ON in the linkage table entry and DAT mode is being changed, AMDSASVI relocates the caller's register 1.		
13 AMDSASVI copies SSAR13 into PG0SR13, SS0RIG14 into PG0SR14, and SSPSW into the SVC old PSW FLCSOPSW.		
14 AMDSASVI adds an entry to the trace table, using the return SVC number and the original SVC old PSW.		

Diagram SADMP-42. AMDSASVI–SVC Interruption Handler (Part 7 of 8)

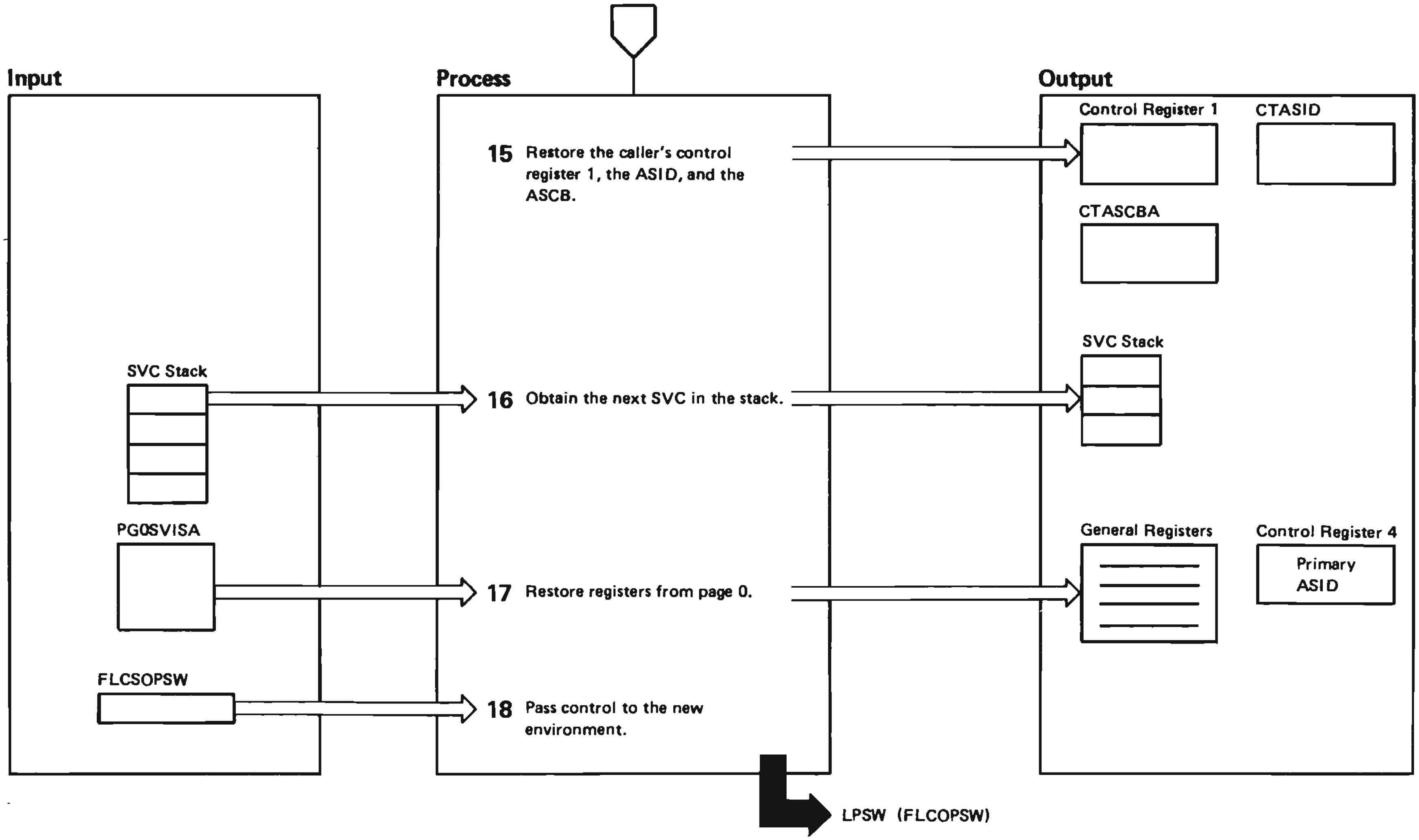


Diagram SADMP-42. AMDSASVI–SVC Interruption Handler (Part 8 of 8)

Extended Description	Module	Label
15 AMDSASVI loads control register 1 from SSSTDCR1, then copies SSASID into CTASIDA and SSASCB into CTASCB.		
16 AMDSASVI decreases SSINDEX to obtain the next SVC in the stack.		
17 AMDSASVI copies CTASIDA into control register 4 and loads all 16 general registers from PGOSVISA.		
18 AMDSASVI loads the modified SVC old PSW FLCSOPSW.		

Diagram SADMP-43. AMDSATER–Virtual Storage Dump Tape Error Recovery (Part 1 of 2)

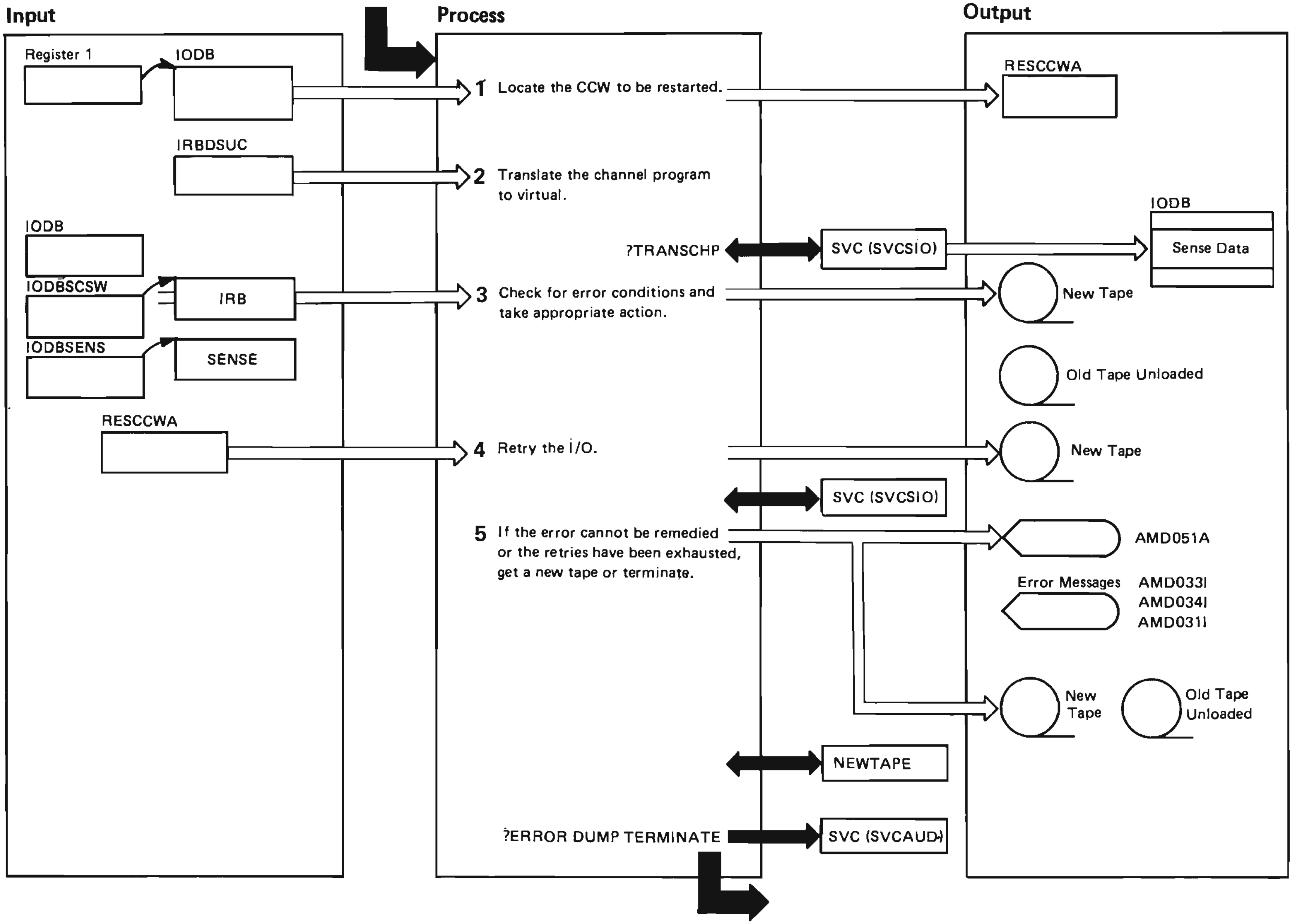


Diagram SADMP-43. AMDSATER — Virtual Storage Dump Tape Error Recovery (Part 2 of 2)

Extended Description	Module	Label
1 AMDSATER locates the CCW to be restarted. IODBCCWA contains the address of the channel program to be restarted. If the CCW to be restarted is data chained, AMDSATER decreases the address of the CCW by the length of a CCW to obtain the starting CCW that failed. IODBSCSW also contains the IRB status information.		
2 AMDSATER translates the channel program to be restarted from real to virtual addressing.		
3 The IRB status (IODBCSW) and the sense data pointed to by IODBSENS define the error and the appropriate action.		
4 The CCWA contains the address of the CCW that is to be retried. AMDSATER calls AMDSASIO (via an SVC macro) to retry the I/O, and executes steps 1 - 3 until the I/O is successful or until the retry count is exhausted. AMDSATER issues the recovery retry I/O and does not wait for the I/O to complete. If an error recurs, a new call to AMDSATER handles the error. AMDSATER is disabled, so no recursion can exist.		
5 If the error is uncorrectable, or if the number of retries is exhausted, AMDSATER attempts to recover by obtaining a new output tape. AMDSATER passes control to entry point AMDSANTP via an SVC if the error is a data check, equipment check, a file protection exception, an intervention required, a loadpoint error or end-of-reel condition. If recovery fails, or if AMDSATER cannot obtain a new tape, AMDSATER issues message AMD033I to the console. If SENSE data is available, AMDSATER also issues message AMD034I. If AMDSATER has no output tape left to write to, it issues message AMD031I and terminates SADMP.		

Diagram SADMP-44. NEWTAPE--Subroutine of AMDSATER (Part 1 of 2)

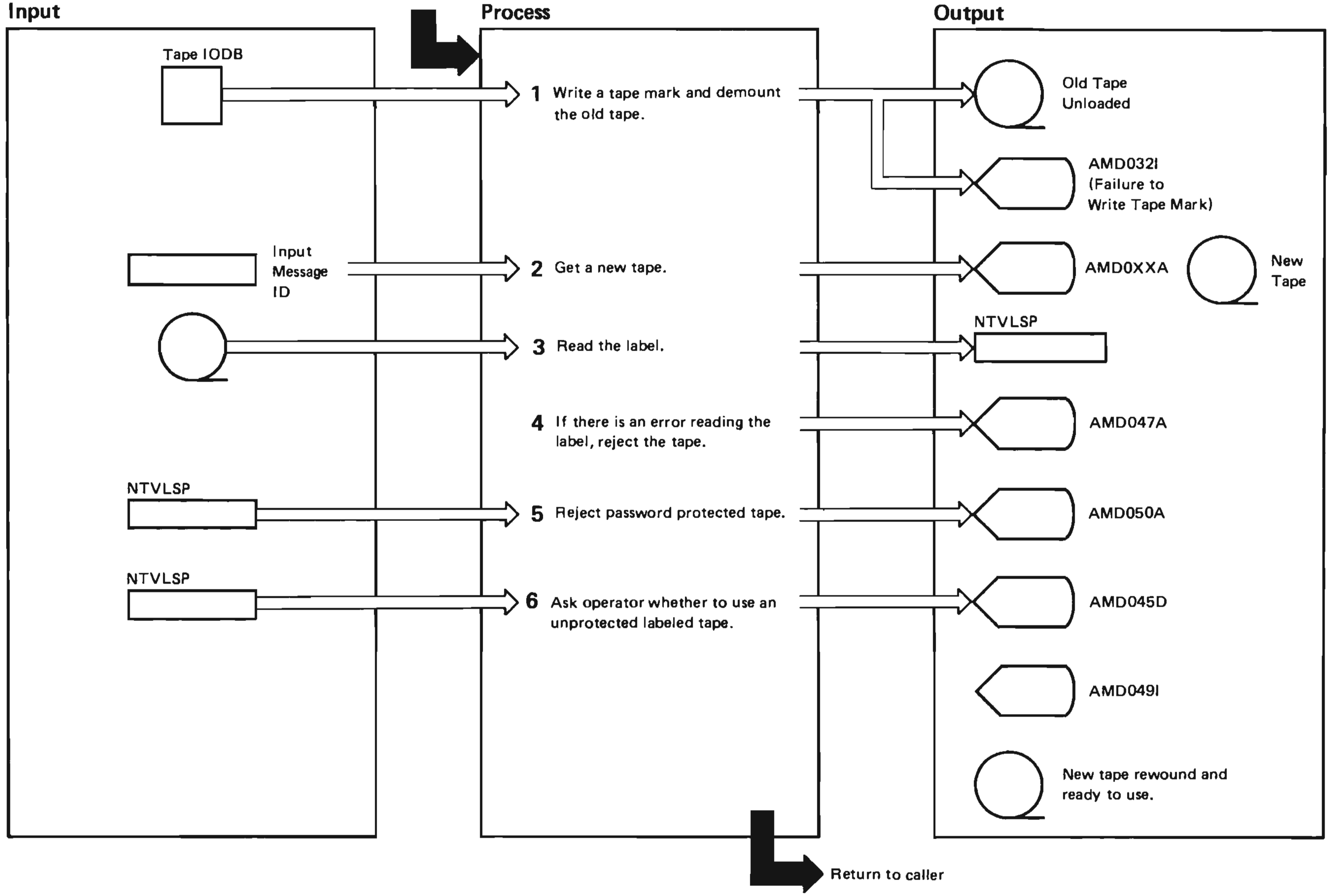


Diagram SADMP-44. NEWTAPE–Subroutine of AMDSATER (Part 2 of 2)

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

1 AMDSANTP writes a tape mark to the old output tape. If the write tape mark fails, AMDSANTP issues message AMD032I. AMDSANTP then rewinds and unloads.

2 AMDSANTP issues message AMD019A, AMD004A, or AMD051A to request mounting of a new tape. If the output tape is a 3480 tape drive, AMDSANTP writes an appropriate display message to the tape drive. Then AMDSANTP waits for the tape to be loaded.

3 AMDSANTP reads the label on the tape. If the tape has no label, AMDSANTP accepts the tape.

4 If the tape has a label that cannot be read, AMDSANTP rejects the tape and issues message AMD050A to request mounting of another tape. If the output tape is a 3480 tape drive, AMDSANTP writes an appropriate display message to the tape drive.

5 If the tape is password protected, AMDSANTP rejects the tape and issues message AMD047A to request mounting of another tape. If the output tape is a 3480 tape drive, AMDSANTP writes an appropriate display message to the tape drive.

6 If the tape has a label that can be read, and the tape is not password protected, NEWTAPE issues message AMD045D to ask the operator whether to use the labeled, unprotected tape. If the operator does not give a correct reply to message AMD045D, NEWTAPE issues message AMD049I to inform the operator of the syntax error.

If the operator rejects the tape, NEWTAPE asks the operator to mount another tape, and repeats steps 3-6. If the operator accepts the tape, NEWTAPE returns to the caller.

See Diagram 45. AMDSAT80 — Virtual Storage Dump.

Diagram 45. AMDSAT80—Virtual Storage Dump Tape Error Recovery for 3480 Tape Drives (Part 1 of 6)

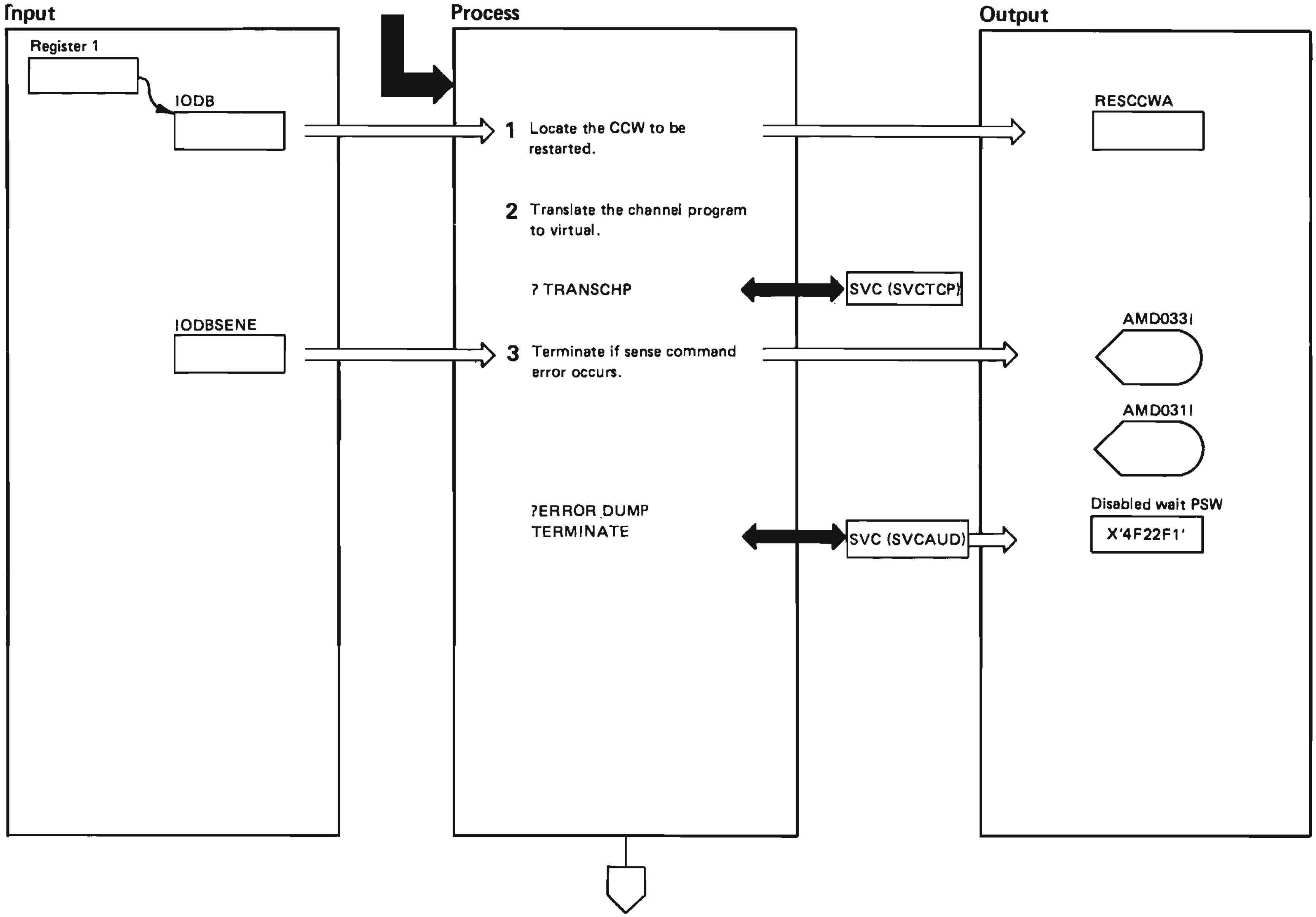


Diagram 45. AMDSAT80–Virtual Storage Dump Tape Error Recovery for 3480 Tape Drives (Part 2 of 6)

Extended Description	Module	Label
1 AMDSAT80 locates the CCW to be restarted. IODBCCWA contains the address of the channel program to be restarted. If the CCW to be restarted is data-chained, AMDSAT80 decreases the address of the CCW by the length of a CCW to obtain the starting CCW that failed.		
2 AMDSAT80 translates the channel program to be restarted from real to virtual addressing.		
3 If an error occurred on the sense command issued previously by the I/O services, AMDSAT80 writes error messages AMD033I and AMD031I and terminates SADMP.		

Diagram 45. AMDSAT80–Virtual Storage Dump Tape Error Recovery for 3480 Tape Drives (Part 3 of 6)

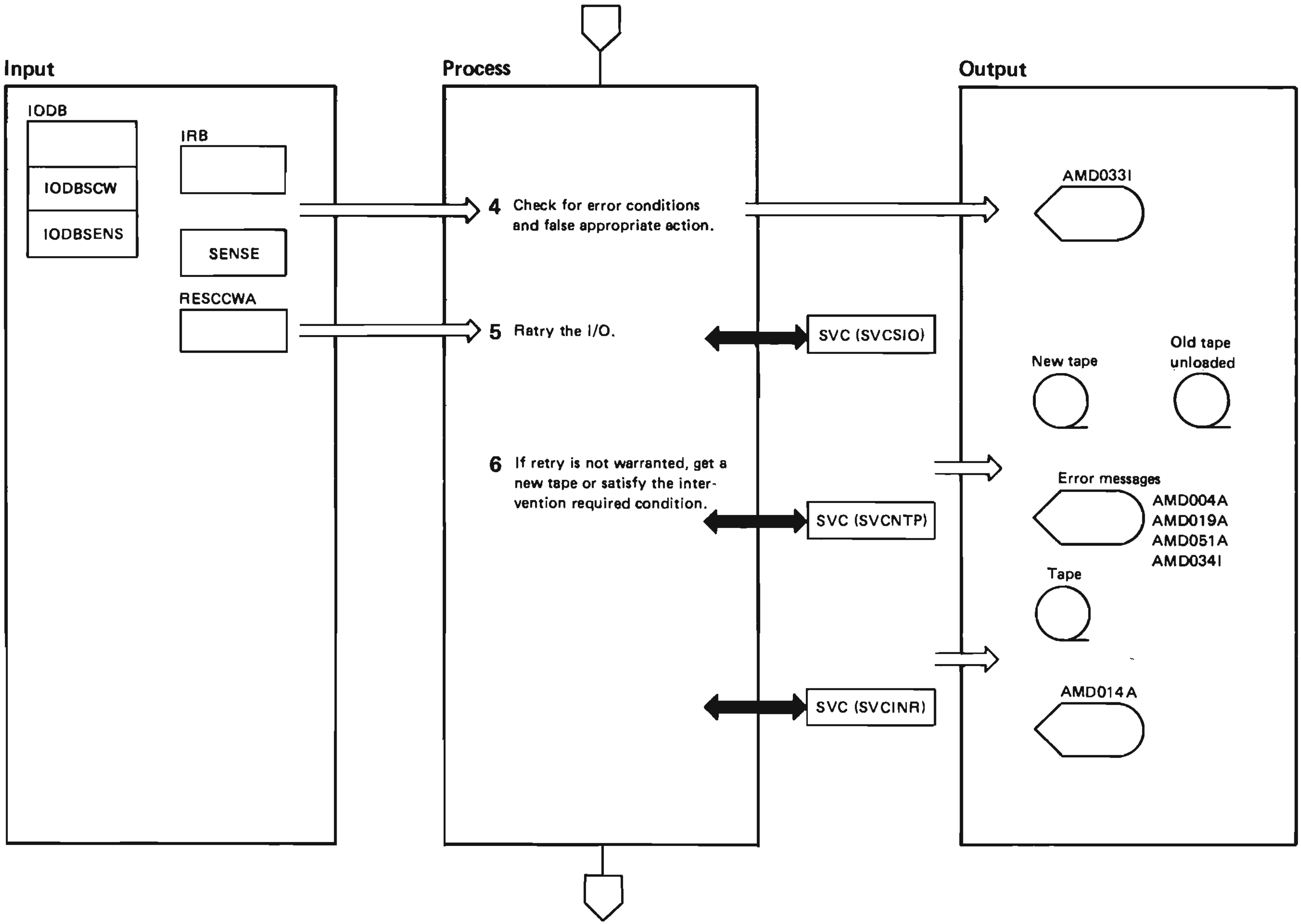


Diagram 45. AMDSAT80–Virtual Storage Dump Tape Error Recovery for 3480 Tape Drives (Part 4 of 6)

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

4 The sense data and the IRB status pointed to by IODBSCSW define the error and the appropriate action. AMDSAT80 issues message AMD033I to notify the operator what type of I/O error occurred.

5 If the error is a channel data check, a chaining check, a demark data buffer error, an environmental data present error, a block id sequencing error, a degraded mode error, or a log and retry error, AMDSAT80 retries the channel program.

6 If the error is a tape length error, a unit exception, or a file protected error, AMDSAT80 informs the operator via a message to mount a new tape. AMDSAT80 passes control to AMDSANTP via an SVC. AMDSANTP rewinds and unloads the old tape and obtains a new tape.

If the error is an intervention required condition, a drive reset error, a manual unload error, a load failure or a load assistance error, AMDSAT80 informs the operator that intervention is required on the tape drive and passes control to AMDSAINR via an SVC. AMDSAINR handles the intervention required processing.

Diagram 45. AMDSAT80—Virtual Storage Dump Tape Error Recovery for 3480 Tape Drives (Part 5 of 6)

Input

Process

7 If error is not recoverable, terminate.

?ERROR DUMP
TERMINATE

SVC (SVCAUD)

Output

AMD031I
AMD034I

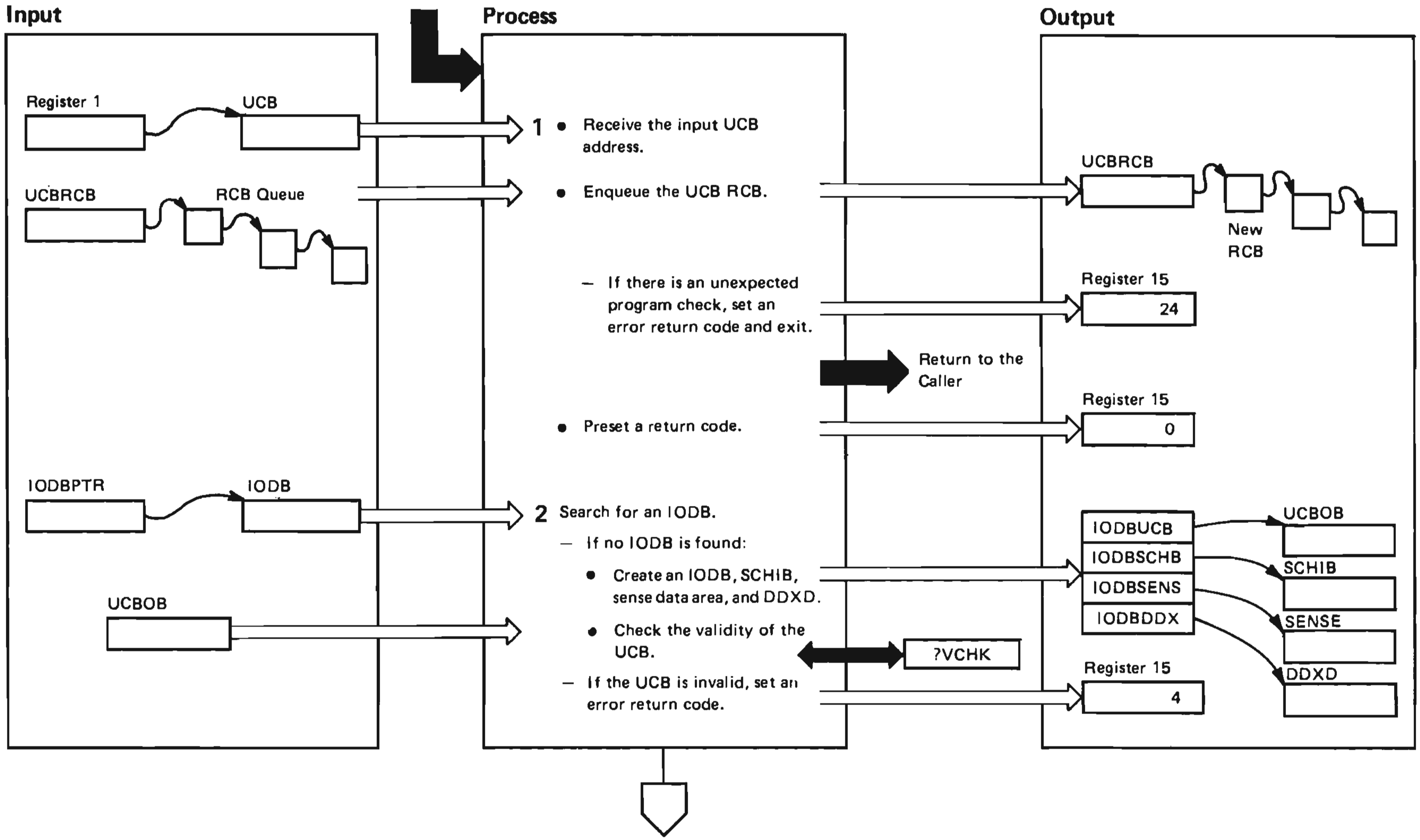
Disabled wait PSW

X'4F22F1'

Diagram 45. AMDSAT80—Virtual Storage Dump Tape Error Recovery for 3480 Tape Drives (Part 6 of 6)

Extended Description	Module	Label
7 For all other errors, AMDSAT80 cannot recover. AMDSAT80 issues error message AMD031I and terminates SADMP.		

Diagram SADMP-46. AMDSAUCB–UCB Search and IODB Update (Part 1 of 6)



Extended Description	Module	Label
AMDSAUCB converts a UCB address into an IODB address for the macro interface ?IO OPEN.		
1 AMDSAUCB sets up the UCB base map, enqueues a UCB RCB, and presets a return code of 0. If AMDSAUCB encounters an unexpected program check when it tries to enqueue the UCB RCB, AMDSAUCB sets an error return code of 20 and exits.		
2 AMDSAUCB searches the queue for an IODB to match the UCB. If AMDSAUCB does not find a matching IODB, AMDSAUCB creates an IODB, and a corresponding SCHIB, sense data area, and a DDXD, then calls ?VCHK to check the validity of the UCB. If the UCB is invalid, AMDSAUCB sets an error return code of 4.		?VCHK

Diagram SADMP-46. AMDSAUCB–UCB Search and IODB Update (Part 3 of 6)

Input	Process	Output
IODBSCH	**3** Check subchannel ID.	
	– If the subchannel ID is invalid, set an error return code.	Register 15: 8
Register 1 → IODBSCH	**4** Enable the subchannel.	
	– If the subchannel is not operational, set an error return code.	Register 15: 12
SCHV	**5** Check the subchannel device number.	
SCHV: 0	– If the subchannel device number is invalid, set an error return code.	Register 15: 16

Diagram SADMP-46. AMDSAUCB—UCB Search and IODB Update (Part 4 of 6)

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

3 AMDSAUCB checks the subchannel ID. If the subchannel is invalid, AMDSAUCB sets an error return code of 8.

4 AMDSAUCB enables the subchannel. If the subchannel is not operational, AMDSAUCB sets an error return code of 12.

5 AMDSAUCB checks SCHV, the subchannel device number. If the SCHV is invalid, AMDSAUCB sets an error return code of 16.

Diagram SADMP-46. AMDSAUCB–UCB Search and IODB Update (Part 5 of 6)

Input

IODBNG | IODBUNAV

IODBNG 1 or IODBUNAV 1

UCBRCB | RCB Queue

Process

6 Verify that the device can be used.

- If the IODB is invalid, sets fields and issues a message.

?IO

- If an IODB is found, check if device can be used. If the device cannot be used, set an error return code.

7 If this device is a paging storage device and the current UCB is for the base exposure, call AMDSAUCB to get the IODB for the paging exposure and attach it to the base IODB.

8 Dequeue the UCBRCB, return the IODB address, and return to the caller.

SVC(SVCUCB)

Output

IODBNG 1 | IODBUNAV 1

MSGID AMD058I

Register 15 20

Paging IODB | Base IODB | IODBBASA | IODBPXPA

DDXD | DDXI

Original RCB Queue

Register 1 | IODB

Diagram SADMP-46. AMDSAUCB—UCB Search and IODB Update (Part 6 of 6)

Extended Description	Module	Label
6 AMDSAUCB attempts to open the device that the IODB represents. If the IODB is invalid and the device cannot be opened, AMDSAUCB indicates that the IODB is invalid and marks the IODB as unavailable. AMDSAUCB calls ?IO to issue error message AMD058I. If AMDSAUCB finds a matching IODB for the UCB, AMDSAUCB verifies that the device can be used. If the device cannot be used, AMDSAUCB sets an error return code of 20 and exits.		?IO
7 The device is a paging storage device if it is a multiple-exposure 3350 (UCBTBYT4 = DT3350 and UCBMTPXP is on). AMDSAUCB turns on IODBPGST to indicate that the device is a paging storage device. AMDSAUCB creates the DDXI.		
8 AMDSAUCB dequeues the UCB RCB, returns the IODB address in register 1, and returns to the caller. If the device is a paging storage device, the IODB returned is for a paging exposure.		

Diagram SADMP-47. AMDSAUPD—Page-In Management (Part 1 of 6)

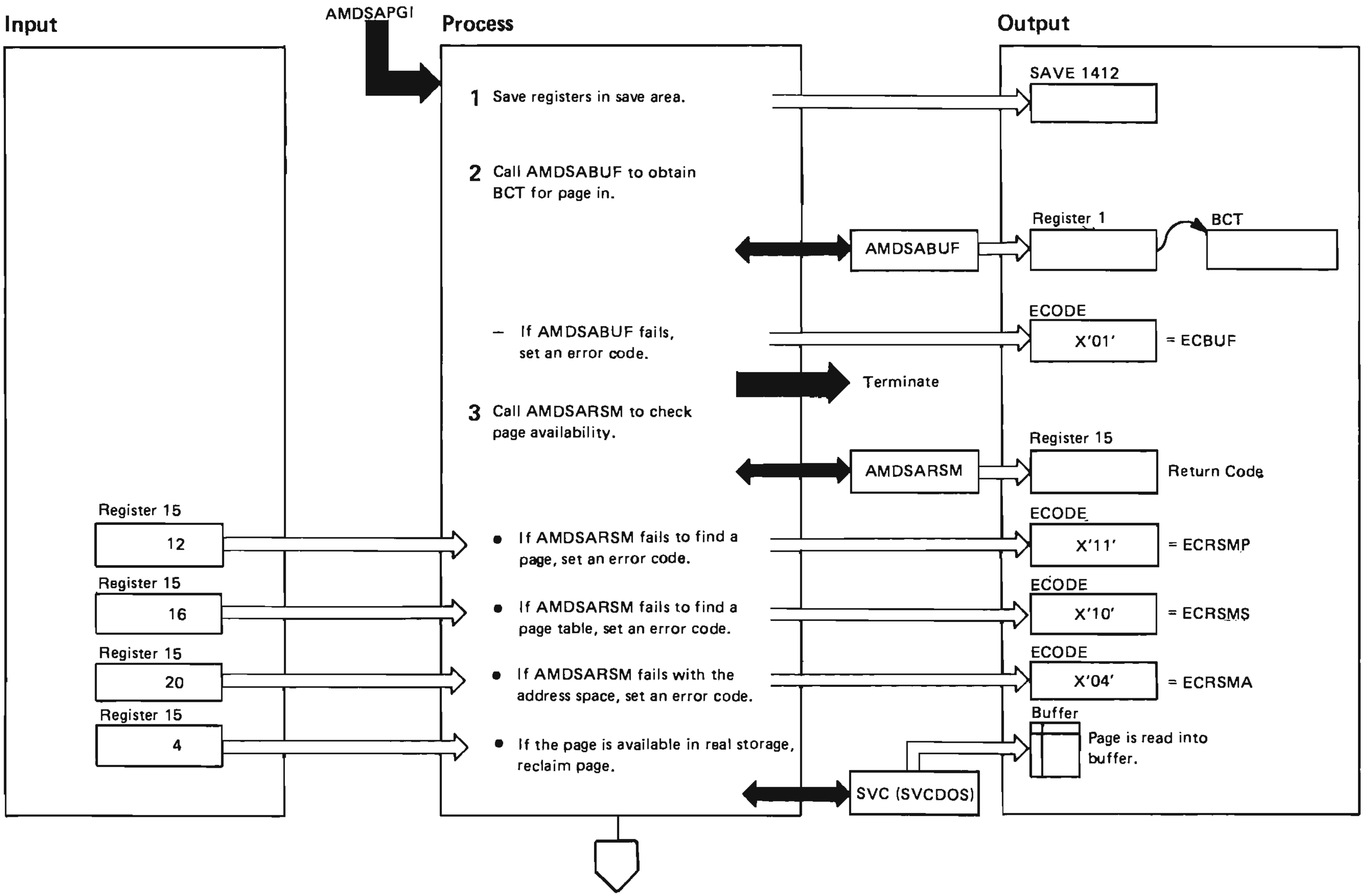


Diagram SADMP-47. AMDSAUPD — Page-In Management (Part 2 of 6)

Extended Description	Module	Label
1 AMDSAUPD saves the entry registers in the save area provided by AMDSASVI.		
2 AMDSAUPD calls AMDSABUF to obtain a BCT for a page in. If AMDSABUF fails to get a BCT, AMDSAUPD sets the error code to X'01' and terminates.		
3 AMDSAUPD calls AMDSARSM to check if the page is available. If AMDSARSM fails to find the page, AMDSARSM sets a return code of 12 and AMDSAUPD sets the error code to X'11'. If AMDSARSM fails to find the page table, AMDSARSM sets a return code of 16 and AMDSAUPD sets the error code to X'10'. If AMDSARSM fails with the total address space, AMDSARSM sets a return code of 20 and AMDSAUPD sets the error code to X'04'. If AMDSARSM determines that the page is available in real storage, it sets a return code of 4 and AMDSAUPD reclaims the page. AMDSAUPD reclaims the page by calling AMDSADOS via SVC to copy the page into the buffer.		
If AMDSARSM has paged the page in, AMDSARSM sets a return code of 0. In this case, AMDSAUPD does nothing to bring the page in.		

Diagram SADMP-47. AMDSAUPD–Page-In Management (Part 3 of 6)

Input

Register 15: 4

Register 15: 0

Process

- If the page is on auxiliary storage, call AMDSAASM to get IODB.

AMDSAASM

- If AMDSAASM fails, set an error code.
- If AMDSAASM succeeds, read the page.
  - If unable to read the page into the buffer, set an error code.

Output

Register 15

ECODE: X'05' = ECASM

Buffer: Page is read into buffer.

ECODE: X'06' = ECREAD

Diagram SADMP-47. AMDSAUPD—Page-In Management (Part 4 of 6)

Extended Description	Module	Label
If AMDSARSM determines that the page is on auxiliary storage, AMDSARSM sets a return code of 4. AMDSAUPD then calls AMDSAASM to get the IODB address for the page. If AMDSAASM fails, it sets a nonzero return code, and AMDSAUPD sets the error code to X'05'. If AMDSAASM succeeds, AMDSAASM sets a return code of 0 and AMDSAUPD reads the page into the BCT. If AMDSAUPD cannot read the page into the BCT, AMDSAUPD sets the error code to X'06'.		

Diagram SADMP-47. AMDSAUPD—Page-In Management (Part 5 of 6)

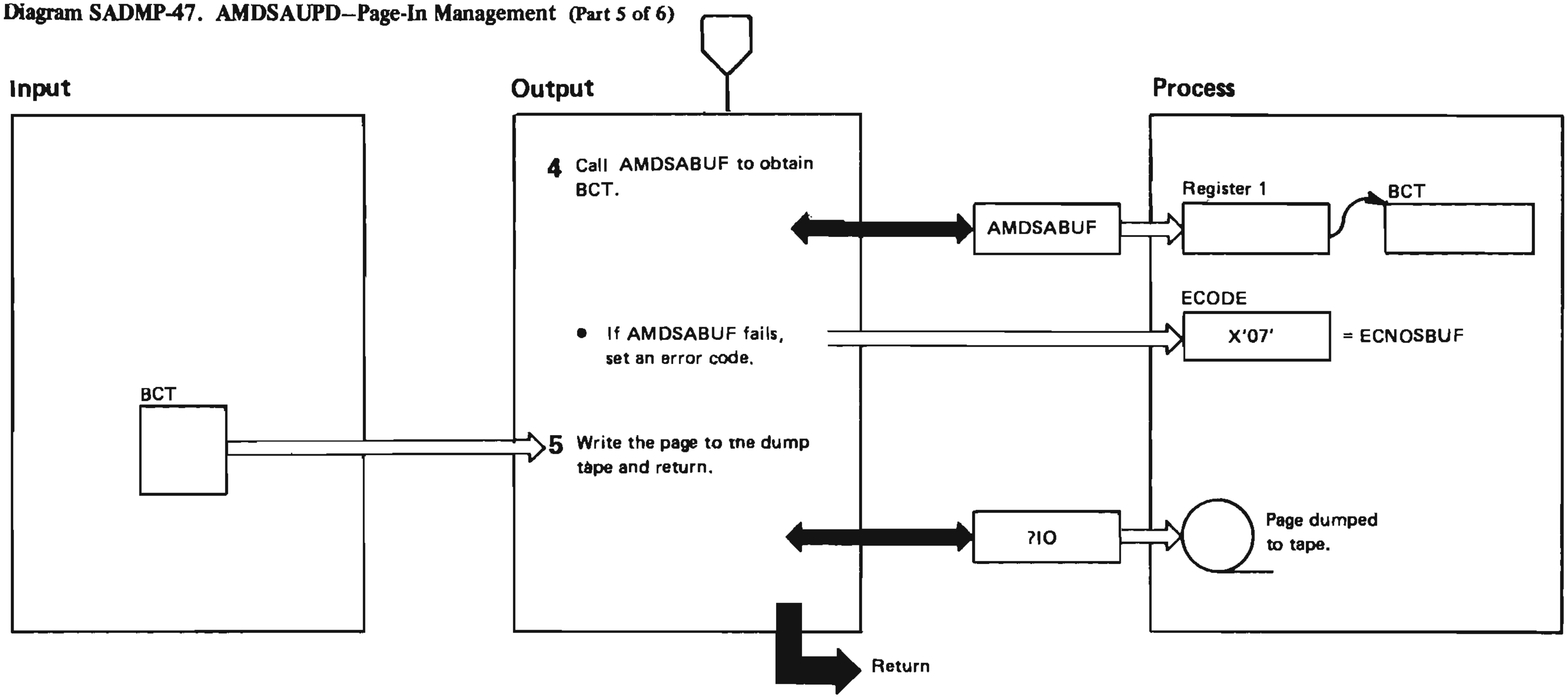


Diagram SADMP-47. AMDSAUPD—Page-In Management (Part 6 of 6)

Extended Description	Module	Label
4 AMDSAUPD calls AMDSABUF to obtain a buffer. If AMDSABUF fails, AMDSAUPD sets the error code to X'07'.		
5 AMDSAUPD calls ?IO to dump the page to tape. AMDSAUPD returns to the previous save area. The return linkage of AMDSAUPD is via AMDSASVI to the point of interruption where AMDSAPGI originally had control.		

Diagram SADMP-48. AMDSAVCK–Virtual Storage Dump Validity Check (Part 1 of 4)

Input

VCKRCB

RCB Queue

CBER018 = 'INVALID'

CBER018 = 'MISSING'

Process

1 Save the ID and address of the control block to be checked.

2 Enqueue the validity check RCB.

3 Perform a validity check appropriate to the control block ID.

- For an unknown control block ID, dequeue the validity check RCB from the RCB queue and call for a dump.

?ERROR REASON (ECUNKN) DUMP RECOVER

4 Set a return code for failed validity checks and issue a message for invalid control blocks.

?IO MSGID (AMD018I)

Output

Register 0

Register 1

CBADDR

VCKRCB

New RCB

Original RCB Queue

Register 15

4

Diagram SADMP-48. AMDSAVCK — Virtual Storage Dump Validity Check (Part 2 of 4)

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

1 AMDSAVCK saves the ID and address of the control block to be checked.

2 AMDSAVCK enqueues the validity check RCB.

3 AMDSAVCK performs the validity check appropriate to the control block ID. The control blocks whose IDs might be checked are:

- the RAB
- ASCB
- ASXB
- ASVT
- EDB
- GTFBCB
- MCCE
- MCQE
- PART
- SGT
- LPMB
- UCB
- ASMVT
- TCB
- SART
- EST

For an unknown control block ID, AMDSAVCK dequeues the validity check RCB from the RCB queue and requests a dump.

4 AMDSAVCK sets a return code of 4 for failed validity checks and issues message AMD018I for invalid control blocks.

Diagram SADMP-48. AMDSAVCK–Virtual Storage Dump Validity Check (Part 3 of 4)

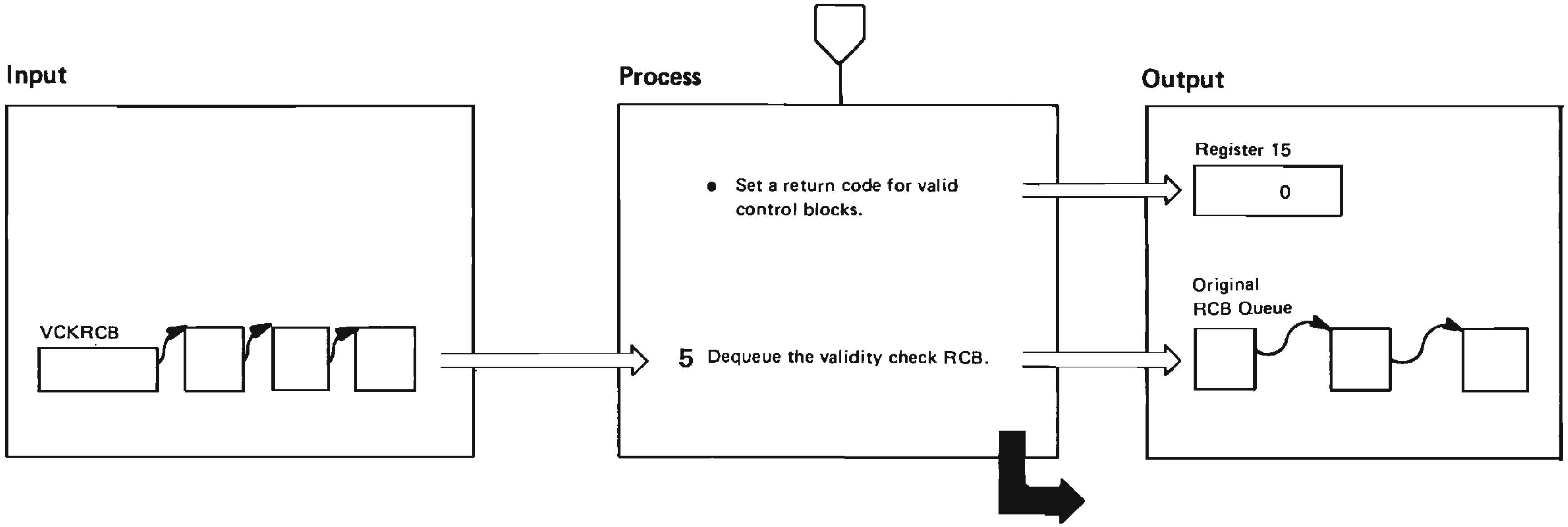


Diagram SADMP-48. AMDSAVCK—Virtual Storage Dump Validity Check (Part 4 of 4)

Extended Description	Module	Label
AMDSAVCK sets a return code of 0 for valid control blocks.		
5 AMDSAVCK dequeues the validity check RCB and exits.		

Diagram SADMP-49. TCBSCAN–Scan TCBs for this Address Space and Dump Associated Subpools (Part 1 of 2)

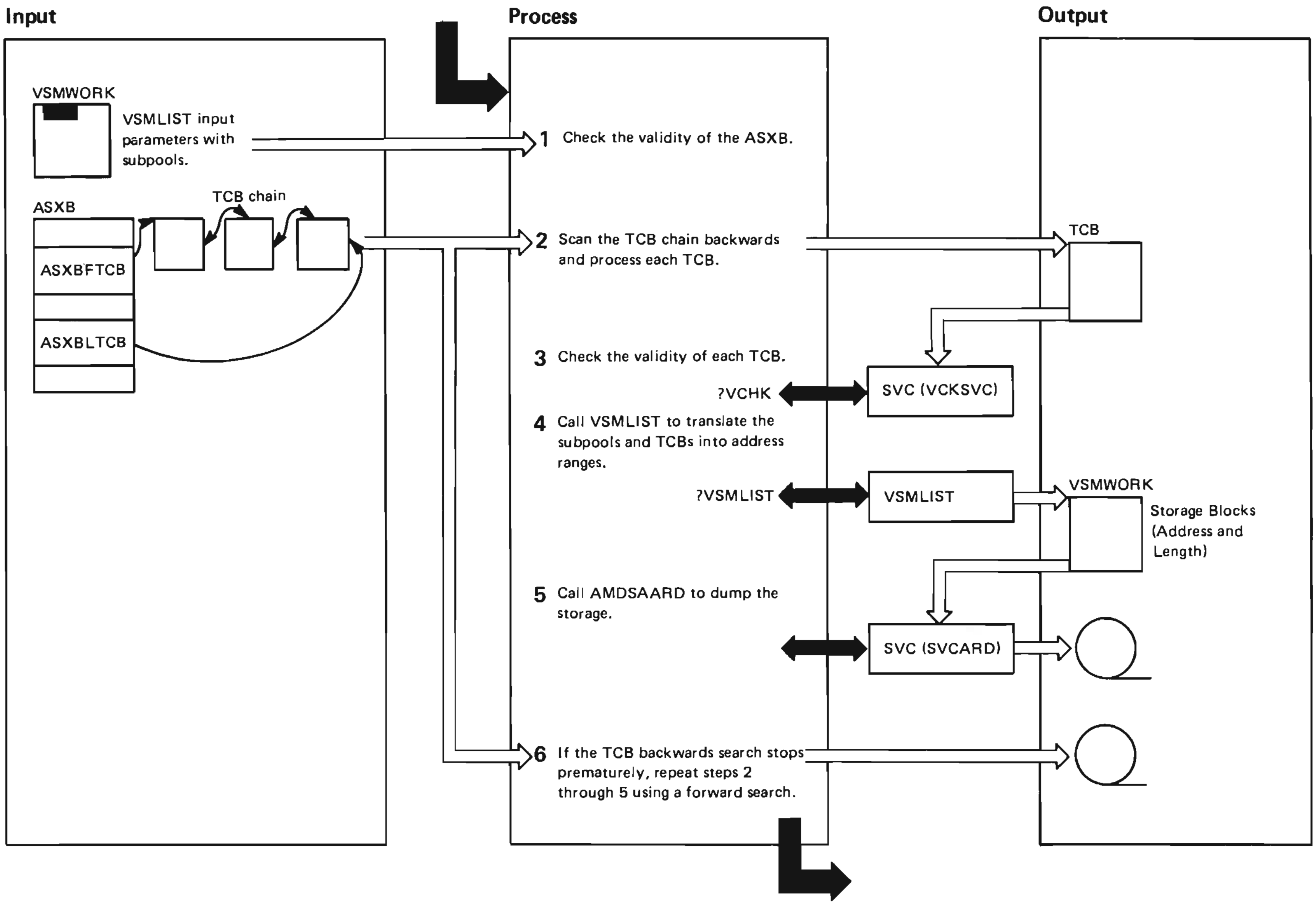


Diagram SADMP-49. TCBSCAN–Scan TCBs for this Address Space and Dump Associated Subpools (Part 2 of 2)

Extended Description	Module	Label
1 TCBSCAN calls AMDSAVCK to check the validity of the ASXB for this address space.		
2 TCBSCAN scans the TCB chain backwards, and processes each TCB. The head of the backward TCB chain is ASXBLTCB and the chain field is TCBBACK.		
3 TCBSCAN calls AMDSAVCK to check the validity of each TCB.		
4 TCBSCAN invokes the VSMLIST macro to translate into address ranges all the subpools that are to be dumped in this address space that are also associated with this TCB.		
5 TCBSCAN calls AMDSAARD to accumulate and dump the address ranges.		
6 If TCBBACK is 0 during the backward scan, the TCB queue is broken. Rescan the TCB queue in the forward direction by beginning from ASXBFTCB and chaining from TCBTCB. TCBSCAN repeats steps 2–5.		

Method of Operation for the Logic Tool material

This section has detailed information for AMDSADMP modules. These modules are in alphabetical order. The information is broken down into four separate headings. The four headings and the topics they include are:

Module Description,:

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

Note: These modules are also included in the *System Logic Library*.

Module Operation:

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

Diagnostic Aids, which provide information useful for debugging program problems:

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

Logic Diagram, which illustrates the processing of the module, the input it uses, the output it produces, and the flow of control. Some modules do not have a logic diagram because the processing is sufficiently explained in the module description, module operation, and diagnostic aids sections. Figure 4-2 on page 4-207 illustrates the graphic symbols and format used in the logic diagram.

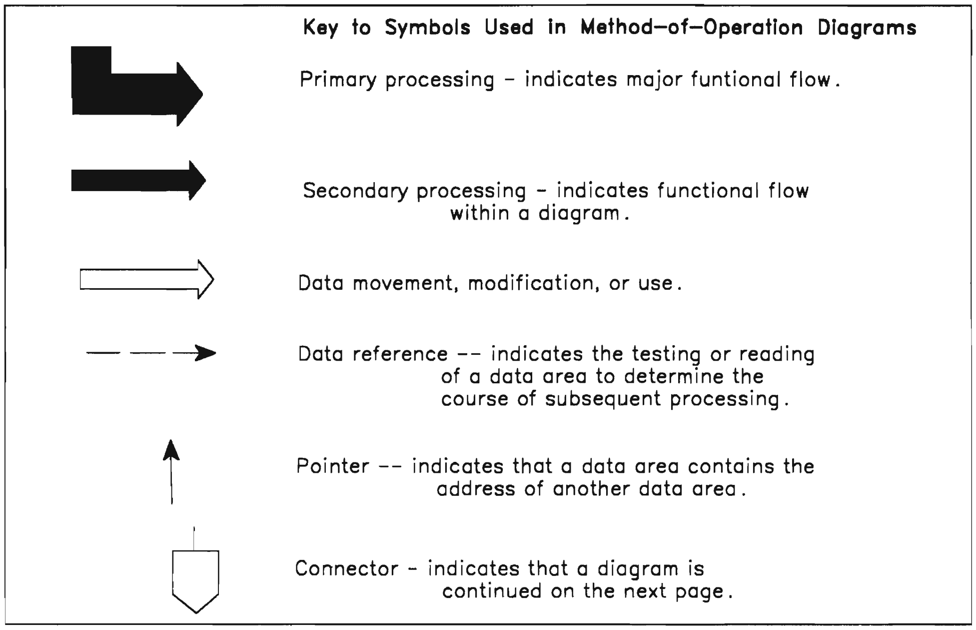


Figure 4-2. Graphic Symbols and Format Used in the Logic Diagram

AMDSADCM - MODULE DESCRIPTION

DESCRIPTIVE NAME: Console Message Dump.

FUNCTION:
Writes virtual dump console messages to the output tape.

ENTRY POINT: AMDSADCM.

PURPOSE: See FUNCTION.

LINKAGE: BALR.

CALLERS: None

INPUT:
```
   Standard parameter list:
     1. DCMIFUNC - function code (fullword).
          - OPEN  (DCMIFUNC=DCMFOPEN)
          - PUT   (DCMIFUNC=DCMFPUT)
          - WRITE (DCMIFUNC=DCMFWRIT)
     2. DCMITYPE - message type (PUT only, 1 byte).
     3. DCMILEN  - message length (PUT only, 1 byte
                   positive integer).
     4. DCMITEXT - message text (PUT only).
```

OUTPUT:
```
   OPEN -  AMDSADCM marks the message dump service ready
           for use or unusable.

   PUT  -  If the message length is zero, AMDSADCM
           ignores the request.  If there is a current
           buffer available, AMDSADCM places the message
           into the message buffer, and writes all full
           buffers to the output tape.  If there are no
           buffers available, AMDSADCM increments the
           count of discarded messages and sets the
           module return code to 4.  Note:  Messages
           longer than 79 characters are truncated to 79
           characters.

   WRITE - If the output tape is usable, AMDSADCM writes
           all buffered messages to tape.  If an error
           occurs while attempting to use the output tape,
           AMDSADCM marks the message dump service
           unusable and sets the module return code to 4.
```

EXIT NORMAL: To caller via BR 14.

EXIT ERROR: To caller via BR 14.

EXTERNAL REFERENCES:

ROUTINES:
```
   AMDSAGTM (SVC) - Gets and frees storage.
   AMDSASIO (SVC) - Writes a message buffer to the output tape.
   AMDSABUF (SVC) - Gets buffers.
   AMDSASVI (SVC interrupt handler)
     For SVC modules with name AMDSA---, linkage
     is by SVC(SVC---) or SVC(---SVC).
```

CONTROL BLOCKS:
```
                C=created, R=referenced, M=modified, D=deleted
                All are private to Stand-Alone Dump.
   CCT    R    - Common Control Table
   RCB    CRMD - Recovery Control Block
   IODB   R    - I/O Device Block
   DCM    CRM  - Console Message Dump descriptor
   DCMD   CRMD - Console Message Dump Data
```


AMDSADCM - MODULE OPERATION

AMDSADCM performs a different function depending on the input function code (DCMIFUNC):

- OPEN (DCMIFUNC=DCMFOPEN): Initializes the console message dump service by performing the following steps:

-- 1. Initializes the console message dump descriptor (DCM).

-- 2. Obtains storage for message buffers and puts the addresses of the buffers in the DCM.

-- 3. Initializes the buffers.

-- 4. Sets the "open" indicator, DCMOPEN, to 1.

-- 5. Resets the "unavailable" indicator, DCMUNAV, to 0.

- PUT (DCMIFUNC=DCMFPUT): Processes the input message parameters:

DCMICODE indicates the type of message and is passed directly to the tape.
DCMITEXT is the variable-length message text.
DCMILEN is the message length.

Note: The console message dump data mapping DCMD maps the message output buffers, each of which holds approximately 50 messages of 79 characters each.

PUT processing performs the following steps:

-- 1. If the console message dump service is functional, and the input message text has nonzero length, AMDSADCM places the message into the current buffer.

Note: Messages longer than 79 characters are truncated to 79 characters.

-- 2. If there is no current buffer, AMDSADCM attempts to obtain a new buffer. If there are no more buffers, AMDSADCM increments the count of discarded messages. If there is no more room in the message dump address space, AMDSADCM indicates this in the buffer (DCMDSMAX=1), and marks the console message dump service unavailable (DCMUNAV=1).

-- 3. If the current buffer is full, AMDSADCM marks the buffer full, and writes full buffers to the output tape.

-- 4. If the console message dump service is not functional, AMDSADCM sets the module return code to 4.

PUT cannot guarantee that every message will appear in the dump output because it must postpone message writing during tape error recovery. It is possible (but not likely) that error recovery may generate enough messages to fill up all available buffers. Furthermore, the message dump address space is of finite size.

AMDSADCM - MODULE OPERATION (Continued)

Finally, PUT cannot write messages to a tape that has been rewound.

In the absence of SADMP internal errors, AMDSADCM will almost always have enough buffer space to dump any messages except those that appear after the dump output device has been logically closed.

- WRITE (DCMIFUNC=DCMFWRIT): Performs the following steps:

-- 1. Changes the status of the current buffer to full.

-- 2. Attempts to write all full buffers to the output tape.

Buffer State Transition Diagram

```
     +------------+   GETCURR    +------------+
+--->|   Empty    |------------->|  Current   |----+
|    | (DCMSEMPT) |              | (DCMSCURR) |    |
|    +------------+              +------------+    |
|                                                  |
| WRITEALL                                         | FULLBUFF
| (I/O successful)                                 |
|                                                  |
|    +------------+   WRITEALL   +------------+    |
+----|   Write    |<-------------|    Full    |<---+
     | (DCMSWRIT) |------------->| (DCMSFULL) |
     +------------+   WRITEALL   +------------+
                    (I/O failed)
```

RECOVERY OPERATION:

Marks the console message dump service unusable by setting DCMUNAV=1 and exits with return code 4.

AMDSADCM - DIAGNOSTIC AIDS

ENTRY POINT NAME: AMDSADCM.

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

AMDSADCM returns a 0 in register 15.

EXIT ERROR:

AMDSADCM returns a 4 in register 15.

REGISTER CONTENTS ON ENTRY:

R1 - Points to standard parameter list.
R13 - Points to caller's save area.
R14 - Return address.
R15 - Entry point address.

REGISTER CONTENTS ON EXIT:

EXIT NORMAL:

R0 thru R14 - same as at entry.
R15 - Contains return code.

EXIT ERROR:

R0 thru R14 - same as at entry.
R15 - Contains return code.

Diagram 50. AMDSADCM Stand-Alone Dump Console Message Dump

AMDSADCM - Console Message Dump. STEP 01

AMDSADCM

PARAMETERS

DCMIFUNC DCMITYPE
DCMILEN DCMITEXT

Writes virtual dump console messages to the output tape.

01 Sets up the recovery environment.

AMDSARMK
SAVENAME

PARAMETERS
DCMIFUNC

02 If the "open" function is requested, performs OPEN processing.

AMDSARMK
DCMFOPEN

OPEN: 08

03 If the "put" function is requested, performs PUT processing.

AMDSARMK
DCMFPUT

PUT: 11

04 If the "write" function is requested, performs WRITE processing.

AMDSARMK
DCMFWRIT

WRITE: 16

AMDSARMK
RCDCMERR

05 If the request is not valid, sets the module return code to 4.

06
DCMEXIT

06 Normal and abnormal exit: Deletes the recovery environment. Returns to the module caller.

RECOVER
(DELETE, DCMRCB)

AMDSARMK
ON RCDCMERR

ERREXIT

07 Error exit: Sets the module return code to 4, sets the "DCMUNAV" indicator to 1 to indicate that the message dump service is unavailable, and goes to normal exit.

AMDSADCM - Console Message Dump. STEP 08

>DCMEXIT: 06

AMDSARMK
DCMENDA DCMD

08 OPEN

AMDSARMK
RCDCMERR

08 Opens the console message dump service.

09 Gets and initializes message dump buffers.

SVC SVCBUF

AMDSARMK
R1

AMDSARMK
BCTPGTEP
BCTSGTEP
BCTKEY1
BCTKEY2
BCTASID
DCMDHDR

AMDSARMK
RCREG

AMDSARMK
R1 ON

AMDDATA
PRDASCMD

10 Returns to the subroutine caller.

11 PUT

11 Puts a message in the console message dump.

PARAMETERS
DCMILEN

12 If the message service is functional and message text exists, processes the message.

A. If there is no current buffer, attempts to obtain a buffer.

GETCURR: 28

AMDSARMK
RCDCMERR

B. If unable to obtain a buffer, increments the lost message count, and sets the module return code to 4.

C. If there is a current buffer and messages were suppressed because of temporary lack of buffer space, accounts for lost messages.

AMDSARMK
DCMDLOST

AMDSADCM - Console Message Dump.

STEP 12D

AMDSARMK

DCMDTEXT

PARAMETERS

DCMILEN

AMDSARMK

DCMDNUM

PARAMETERS

DCMITYPE DCMITEXT

AMDSARMK

DCMDNUM DCMDMSG

D. Copies message to buffer.

E. If buffer is full, marks buffer full.

FULLBUF: 24

AMDSARMK

DCMDNUM
DCMDCODE
DCMDTEXT

13 If the message service is functional, and message text exists, writes full buffers.

WRITEALL: 21

AMDSARMK

RCDCMERR

14 If the message service is not functional, sets the module return code to 4.

15 Returns to the subroutine caller.

16

WRITE

16 Attempts to write the current buffer to the tape.

17 If the message service is functional and there is a current buffer, marks the current buffer full.

FULLBUF: 24

18 If the message service is functional, writes all full buffers.

AMDSADCM - Console Message Dump. STEP 19

WRITEALL: 21

AMDSARMK
RCDCMERR

19 If the message service is not functional, sets the module return code to 4.

20 Returns to the subroutine caller.

21
WRITEALL

21 Attempts to write all full buffers to the tape.

22 Loops through buffers.

AMDSARMK
CTNODUMP

A. If the buffer is full and it is possible to write to the tape, indicates that the buffer is being written, and writes the BCT to the tape.

IO
(XIODBOUT) WRITE PAGE(BCT) WAIT

AMDSARMK
BCTIO RCREG

B. If the write operation was successful, indicates that the buffer is now empty.

C. If the write operation was NOT successful, indicates that the buffer is still full.

23 Returns to the subroutine caller.

Diagram 50. AMDSADCM Stand-Alone Dump Console Message Dump

AMDSADCM - Console Message Dump. STEP 24

24 FULLBUF

24 Changes the status of the current buffer to full, and makes the message dump service unavailable if the message dump address space is full.

25 Marks the current buffer full.

AMDSARMK

DCMENDA DCMD

AMDSARMK

ON

26 If the message dump address space is full, sets the "DCMUNAV" indicator to 1 to indicate that the message dump service is unavailable.

AMDSARMK

DCMDSMAX

27 Returns to the subroutine caller.

28 GETCURR

28 Searches for an empty buffer and makes it the current buffer.

BUFFLOOP

29 Loops through buffers.

AMDSARMK

OFF PAGESIZE

A. If the buffer is empty, initializes the buffer, and makes the buffer the current buffer.

B. If an empty buffer has been found, exits the loop.

AMDSARMK

BCTVADDR
DCMDHDR
DCMDID
DCMDNUM
DCMDSMAX

30 Returns to the subroutine caller.

AMDSAFCM - MODULE DESCRIPTION

DESCRIPTIVE NAME: Stand-Alone Dump Console Message Dump Formatting Module.

FUNCTION:
AMDSAFCM, when invoked by IPCS or AMDPRDMP, formats Stand-Alone Dump messages found in the console message dump.

ENTRY POINT: AMDSAFCM.

PURPOSE: See FUNCTION.

LINKAGE: ATTACH from AMDPRDMP, LINK from IPCS

CALLERS: None

INPUT:
```
   ABDPL    - AMDPRDMP/IPCS parameter list.
   ADPLSYNO - Syntax check operands only.
   ADPLSBPL - Subpool for dynamic storage.
   ADPLBUF  - Address of output buffer.
   ADPLPRNT - Address of print routine.
   ADPLMEMA - Address of memory access routine.
   PRDSADP  - Indicator that a Stand-Alone Dump is being processed.
   PRDASCMD - Console message dump ASID.
   DCMVBEG  - First virtual address used to dump console messages.
   DCMVEND  - Highest virtual address used to dump console messages.
   DCMDNUM  - Index of last message in a DCMD.
   DCMDMSG  - AMDSADMP console message array.
```

OUTPUT:
Formatted messages from console message dump, a message stating that it is not a Stand-Alone Dump being processed, or a message stating that no console message dump messages were found.
Note: These messages do not go to the MVS operator's console.

```
"*** STAND-ALONE DUMP VIRTUAL DUMP MESSAGE LOG ***"

"*** THE DUMP BEING PROCESSED IS NOT A STAND-ALONE DUMP"

"*** THERE ARE NO STAND-ALONE DUMP MESSAGE LOG BUFFERS
     IN THIS DUMP"

"*** ANY FURTHER MESSAGES WERE SUPPRESSED BY STAND-ALONE DUMP
     BECAUSE THE MESSAGE DUMP ADDRESS SPACE WAS FULL"

"*** dddddddddd MESSAGE LOG BUFFER(S) ARE MISSING,
     PROBABLY DUE TO I/O ERRORS DURING THE STAND-ALONE DUMP"

"*** dddddddddd MESSAGE(S) WERE SUPPRESSED BY STAND-ALONE DUMP
     DUE TO A TEMPORARY SHORTAGE OF BUFFER SPACE"
```

EXIT NORMAL: To caller, via BR 14.

EXIT ERROR: Percolation to caller's recovery.

EXTERNAL REFERENCES:

ROUTINES: ADPLESRV - Performs AMDPRDMP/IPCS read and print services.

DATA AREAS:
```
   ADPLBUFR - Buffer containing a line to be printed.
   ADPLVIRT - Indicates a virtual dump record read request.
   ADPLHDR  - Indicates a dump header record read request.
   ADPLASID - ASID for virtual dump record read request.
   ADPLDLEN - Length for virtual dump record read request.
   ADPLPAAD - Dump address to read.
   ADPLPART - Address of AMDPRDMP/IPCS buffer.
```

AMDSAFCM - MODULE DESCRIPTION (Continued)

CONTROL BLOCKS:

C=created, R=referenced, M=modified, D=deleted
#=private to Stand-Alone Dump

#DCMD	R	Console message dump data
ABDPL	RM	Exit parameter list
PRDINPUT	R	Dump title header record

AMDSAFCM - MODULE OPERATION

AMDSAFCM is a verb exit for IPCS and AMDPRDMP, and is invoked by the SADMPMSG control statement. AMDSAFCM processes Stand-Alone Dump messages found in the console message dump. It performs the following steps:

1. AMDSAFCM attempts to read the dump header record, which indicates what type of dump is being processed.

2. If the header can be read, and it indicates that the type of dump being processed is a Stand-Alone Dump, AMDSAFCM prints:

   "*** STAND-ALONE DUMP VIRTUAL DUMP MESSAGE LOG ***"
   and continues processing the dump.

3. If the header can be read, and it indicates that the type of dump being processed is not a Stand-Alone Dump, AMDSAFCM prints:

   "*** THE DUMP BEING PROCESSED IS NOT A STAND-ALONE DUMP"
   and stops processing the dump.

4. AMDSAFCM reads virtual dump records using the console message dump ASID (FFFA) starting with the first virtual address used by AMDSADCM to dump console messages. If the read for a record is successful, AMDSAFCM formats the Stand-Alone Dump messages in the record.

5. AMDSAFCM reads records and formats messages until "NODCMMAX" number of successive reads fail or the highest address used by AMDSADCM to dump messages is reached. This allows for the possibility of some records being missing due to I/O errors without spending excessive time trying to read every record in the permitted range.

6. If any reads fail, AMDSAFCM prints:

   "*** dddddddddd MESSAGE LOG BUFFER(S) ARE MISSING, PROBABLY DUE TO I/O ERRORS DURING THE STAND-ALONE DUMP".

7. If any messages were suppressed, AMDSAFCM prints:

   "*** dddddddddd MESSAGE(S) WERE SUPPRESSED BY STAND-ALONE DUMP DUE TO A TEMPORARY SHORTAGE OF BUFFER SPACE".

8. If the dump header record cannot be read, or this is a Stand-Alone Dump and no DCMDs are found, AMDSAFCM prints

   "*** THERE ARE NO STAND-ALONE DUMP MESSAGE LOG BUFFERS IN THIS DUMP".

   This occurs, for example, when a real dump is taken without a virtual dump.

9. If messages were suppressed due to lack of space in the message dump address space, AMDSAFCM prints:

   "*** ANY FURTHER MESSAGES WERE SUPPRESSED BY STAND-ALONE DUMP BECAUSE THE MESSAGE DUMP ADDRESS SPACE WAS FULL".

RECOVERY OPERATION:

None. If an ABEND occurs and an ABEND dump DD (SYSUDUMP, SYSABEND, or SYSMDUMP) is allocated, MVS will take an ABEND dump.

Diagram 51. AMDSAFCM SADMP Console Message Dump Formatting Module

AMDSAFCM - DIAGNOSTIC AIDS

ENTRY POINT NAME: AMDSAFCM.

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

0

REGISTER CONTENTS ON ENTRY:

R1 - Points to the ABDPL.
R13 - Points to caller's save area.
R14 - Return address.
R15 - Entry point address.

REGISTER CONTENTS ON EXIT: Irrelevant

AMDSAFCM - Stand-Alone Dump Console Message Dump Formatting Module. STEP 01

AMDSAFCM

AMDSAFCM, when invoked by IPCS or AMDPRDMP, formats Stand-Alone Dump messages found in the console message dump.

BLSABDPL

ADPLSYNO

01 If not a syntax check only, prints the header and prints a blank line.

PRINT: 08
HEADER, LENGTH(HEADER)

PRINT: 08
BLANK, LENGTH(BLANK)

AMDSARMK

ON	OFF

02 Tries to read the dump header record to determine whether or not this is a Stand-Alone Dump.

BLSABDPL

ADPLHDR

ACCESS: 11
0, PRDINPTR

AMDDATA

PRDSADP

AMDSARMK

ON	OFF

BLSABDPL

ADPLHDR

A. If this is not a Stand-Alone Dump, prints: "*** THE DUMP BEING PROCESSED IS NOT A STAND-ALONE DUMP".

PRINT: 08
NSADMP, LENGTH(NSADMP)

B. If this seems to be a Stand-Alone Dump, formats and prints the console message dump data (DCMDs).

FORMAT: 04

BLSABDPL

ABDPL

AMDSARMK

R2

03 Returns to the module caller.

AMDSAFCM - Stand-Alone Dump Console Message Dump Formatting Module. STEP 04

AMDSARMK: DCMD
04 FORMAT
AMDDATA: PRDASCMD
AMDSARMK: ON

04 Reads the DCMDs and prints their messages.

BLSABDPL: ADPLASID ADPLDLEN ADPLVIRT

AMDSARMK: DCMD
AMDSARMK: DCMBEGA DCMENDA PAGESIZE

05 Reads and prints the DCMDs until "NODCMMAX" number of successive reads fail or the end of the message dump address space is reached.

ACCESS: 11
BUFADDR, DCMDPTR

A. If the prior reads failed, prints: "*** ddddddddd MESSAGE LOG BUFFER(S) ARE MISSING, *** PROBABLY DUE TO I/O ERRORS DURING THE STAND-ALONE DUMP".

MISSMSG: 22

B. If messages were suppressed because of a temporary lack of buffer space, prints: "*** ddddddddd MESSAGE(S) WERE SUPPRESSED BY STAND-ALONE DUMP *** DUE TO A TEMPORARY SHORTAGE OF BUFFER SPACE".

AMDSARMK: DCMDLOST

BUFSPMSG: 17
DCMDLOST

AMDSARMK: DCMDNUM DCMDMSG
AMDSARMK: DCMDPRNT

C. Converts the messages in a DCMD message array to upper case and prints them.

PRINT: 08
FOLDAREA, LENGTH(FOLDAREA)

AMDSARMK: OCMDSMAX

D. If messages were suppressed because of lack of space in the message dump address space, prints: "*** ANY FURTHER MESSAGES WERE SUPPRESSED BY STAND ALONE-DUMP *** BECAUSE THE MESSAGE DUMP ADDRESS SPACE WAS FULL".

AMDSAFCM - Stand-Alone Dump Console Message Dump Formatting Module. STEP 06

PRINT: 08
BLANK, LENGTH(BLANK)

PRINT: 08
FURSUPP1, LENGTH(FURSUPP1)

PRINT: 08
FURSUPP2, LENGTH(FURSUPP2)

06 If no DCMDs are found in the dump, prints: "*** THERE ARE NO STAND-ALONE DUMP MESSAGE LOG BUFFERS IN THIS DUMP".

PRINT: 08
NODCMDS, LENGTH(NODCMDS)

07 Returns to the subroutine caller.

PARAMETERS

08

LINE LEN PRINT

08 Prints a line using the AMDPRDMP/IPCS print service.

BLSABDPL

ADPLBUFR

09 Prints a line.

ADPLESRV
ABDPL, ADPLSPRT

10 Returns to the subroutine caller.

AMDSAFCM - Stand-Alone Dump Console Message Dump Formatting Module. STEP 11

PARAMETERS

DUMPADDR DATAPTR

11 ACCESS

11 Reads a dump record using the AMDPRDMP/IPCS read service.

BLSABDPL

ADPLPAAD

12 Reads a dump record.

ADPLESRV

ABDPL, ADPLSACC, ADPLPACC

AMDSARMK

RCREG

BLSABDPL

ADPLPART

PARAMETERS

DATAPTR

13 Returns to the subroutine caller.

PARAMETERS

FIXEDIN DECOUT LENOUT

14 DECCONV

14 Converts the data from FIXED(31) to printable decimal, inserts commas, and supresses leading zeros. Output is 'DD,DDD,DDD,DDD' left justified in DECOUT, with the length in LENOUT.

AMDSARMK

R1

PARAMETERS

LENOUT

AMDSARMK

R1

15 Converts to printable decimal.

PARAMETERS

LENOUT DECOUT

16 Returns to the subroutine caller.

AMDSAFCM - Stand-Alone Dump Console Message Dump Formatting Module. STEP 17

PARAMETERS

NUMSUPP

17

BUFSPMSG

17 Prints a message indicating that messages were suppressed by Stand-Alone Dump because of a temporary lack of buffer space.

18 Prints a blank line.

PRINT: 08
BLANK, LENGTH(BLANK)

19 Converts the number of suppressed messages to printable form. Adds this number to the message to be printed.

DECCONV: 14
NUMSUPP, MSGTEXT(MSGPOS1), NUMLEN1

20 Prints the first and second message line and a blank line.

PRINT: 08
MSGTEXT, EVAL(MSGPOS1-1)

PRINT: 08
SUPP2, LENGTH(SUPP2)

PRINT: 08
BLANK, LENGTH(BLANK)

21 Returns to the subroutine caller.

AMDSAFCM - Stand-Alone Dump Console Message Dump Formatting Module. STEP 22

PARAMETERS

NUMMISS

22

MISSMSG

22 Prints a message indicating that the message log buffers are missing probably because of I/O errors during the Stand-Alone Dump.

23 Prints a blank line.

PRINT: 08
BLANK, LENGTH(BLANK)

24 Converts the number of missing messages to a printable form. Puts this number in the message to be printed.

DECCONV: 14
EVAL(NODCMCNT), MSGTEXT(MSGPOS2), NUMLEN2

25 Prints the first and second message line and a blank line.

PRINT: 08
MSGTEXT, EVAL(MSGPOS2-1)

PRINT: 08
BUFMIS2, LENGTH(BUFMIS2)

PRINT: 08
BLANK, LENGTH(BLANK)

26 Returns to the subroutine caller.

AMDSAXSM - MODULE DESCRIPTION

DESCRIPTIVE NAME: Stand-Alone Dump Extended Storage Management.

FUNCTION:

AMDSAXSM executes the PGIN instruction to page data in from extended storage, and may indicate where a page can be found on auxiliary storage.

ENTRY POINT: AMDSAXSM.

PURPOSE: See FUNCTION.

LINKAGE: Branch, standard linkage.

CALLERS: None

INPUT:

```
   BCTBUFRA - Real address of buffer for PGIN.
   RITEST   - Address of the beginning of the EST.
   ESTCHNG  - When 0, a valid copy of the page exists on auxiliary
              storage.
   ESTDATA  - LSID of the auxiliary storage copy of a page on
              extended storage.
```

OUTPUT:

```
   BUFFER   - Frame to which PGIN is done (return code 0 only).
   XSDSPC   - Successful PGIN count (return code 0 only).
   XSDDER   - Data error count (return code 4 or 8).
   XSDBNAC  - Block not available count (return code 4 or 8).
   XSDUEC   - Unexpected error count (return code 8 only).
   BCTLSID  - LSID of page on auxiliary storage
              (return code 4 only).
```

EXIT NORMAL: XSMEXIT - to caller, via BR 14.

EXIT ERROR: XSMEXIT - to caller, via BR 14.

EXTERNAL REFERENCES:

ROUTINES:

```
   AMDSAGTM (SVC) - Get and free automatic data area.
   AMDSABIN (SVC) - Convert to printable hex.
   AMDSACON (SVC) - Write messages to console.
```

CONTROL BLOCKS:

```
   Private to SADMP, in macro AMDSARMK:
     BCT  - Buffer Control Table.
     CCT  - Common Communication Table.
     RCB  - Recovery Control Block (created and deleted).
   MVS control blocks:  None  are modified.
     ESTE, RIT, CVT, PVT, XPTE
```

AMDSAXSM - MODULE OPERATION

AMDSAXSM receives an ESTE (Extended Storage Table Entry) and a BCT (Buffer Control Table). AMDSAXSM attempts to page-in the extended storage frame specified by the ESTE into the buffer associated with the BCT. AMDSAXSM performs the following steps:

1. Sets up an RCB (Recovery Control Block) exit.

2. Checks the validity of the ESTE and the BCT.

3. If the EST (Extended Storage Table) is present and valid, and the ESTE is within the bounds of the EST, AMDSAXSM executes a PGIN instruction to read the page from the extended storage frame into the buffer associated with the BCT.

4. If the PGIN instruction is successful (condition code is 0), AMDSAXSM increments the count of successful PGIN instructions and sets the module return code to 0.

5. If the PGIN instruction fails with a condition code of 1, AMDSAXSM updates the "data error" count and issues message AMD071I.

6. If the PGIN instruction fails with a condition code of 3, AMDSAXSM updates the "block not available" count and issues message AMD071I.

7. If a valid copy of the page exists on auxiliary storage, and the PGIN instruction fails, AMDSAXSM places the LSID (Logical Slot ID) for the auxiliary storage slot where the page resides into the BCT and sets the module return code to 4.

8. If a valid copy of the page does not exist on auxiliary storage, AMDSAXSM sets the module return code to 8.

9. If the EST is not present, or the ESTE is not within the bounds of the EST, AMDSAXSM increments the "unexpected error" count, issues message AMD071I, and sets the module return code to 8.

10. If the EST is present but invalid, AMDSAXSM does not increment any error counts, and does not issue any messages, because message AMD071I was already issued during Stand-Alone Dump initialization.

RECOVERY OPERATION:

If a program check occurs, processing continues at the RCB exit (label: ERROR). This error exit code performs the following steps:

1. Sets the module return code to 8 to indicate that the PGIN operation was not successful.
2. Updates the "unexpected error" count.
3. Issues message AMD071I.

AMDSAXSM - DIAGNOSTIC AIDS

ENTRY POINT NAME: AMDSAXSM.

MESSAGES:

```
AMD071I ERROR IN EXTENDED STORAGE, E-FRAME=ssssssss, DATA ERROR
                                                     BLOCK NOT AVAILABLE
                                   ESTE=eeeeeeee, UNEXPECTED ERROR
```

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

```
0 (RCXSMOK)  - The page-in from extended storage was
               successful.
4 (RCXSMAUX) - The page-in from extended storage failed
               but a valid copy of the page exists on
               auxiliary storage.  The LSID has been
               put into BCTLSID.
8 (RCXSMERR) - The page-in from extended storage failed
               and no valid copy of the page exists on
               auxiliary storage.
```

EXIT ERROR:

8 - The page-in from extended storage failed and no valid copy of the page exists on auxiliary storage.

REGISTER CONTENTS ON ENTRY:

```
R1     - Points to standard parameter list.
R14    - Return address.
R15    - Entry-point address.
```

REGISTER CONTENTS ON EXIT:

EXIT NORMAL:

Registers 0 - 14 restored.
Register 15 contains return code.

EXIT ERROR:

Registers 0 - 14 restored.
Register 15 contains return code.

AMDSAXSM - Stand-Alone Dump Extended Storage Management. STEP 01

AMDSAXSM

PARAMETERS: ESTEPARM BCTPARM

AMDSAXSM executes the PGIN instruction to page data in from extended storage, and may indicate where a page can be found on auxiliary storage.

PARAMETERS: BCTPARM ESTEPARM

01 Initializes.

02 Sets up the recovery environment.

AMDSARMK: SAVENAME

03 Checks the extended storage control blocks for validity.

AMDSARMK: XSDVALID
RIT: RITFESTE RITLESTE

A. If the EST is valid and the ESTE is within the EST, tries a PGIN instruction.

TRYPGIN: 09

AMDSARMK: XSDCURR
AMDSARMK: XSDUEC RCXSMERR

B. If the EST is valid and the ESTE is not within the EST, sets the module return code to 8, updates the "unexpected error" count and issues message AMD071I.

MSG071: 13
ESTE071, ESTEPTR, UNEX071

AMDSARMK: XSDUEC

AMDSARMK: XSDPRES XSDCURR
AMDSARMK: XSDUEC RCXSMERR

04 If the EST is not valid, sets the module return code to 8 to indicate an error on the PGIN instruction. If the EST is not present, updates the "unexpected error" count and issues message AMD071I.

MSG071: 13
ESTE071, ESTEPTR, UNEX071

AMDSARMK: XSDUEC

05 XSMEXIT

05 Normal and abnormal exit: Deletes the recovery environment.

RECOVER
(DELETE, XSMRCB)

AMDSAXSM - Stand-Alone Dump Extended Storage Management. **STEP 06**

AMDSARMK
XSDCURR
AMDSARMK
XSDUEC RCXSMERR

ERROR

06 Returns to the module caller.

07 Error exit: Sets the module return code to 8 to indicate a PGIN failure. Updates the "unexpected error" count and issues message AMD071I.

MSG071: 13
ESTE071, ESTEPTR, UNEX071

08 Goes to the normal exit.

>XSMEXIT: 05

AMDSARMK
XSDUEC

ESTE
09
TRYPGIN
RIT
RITEST
AMDSARMK
BCTBUFRA

09 Attempts to page-in an extended storage frame.

AMDSARMK
XSDCURR
AMDSARMK
XSDSPC RCXSMOK

10 If the PGIN condition code is 0, sets the module return code to 0, and updates the successful PGIN count.

11 If the PGIN condition code is not 0, updates the appropriate error count, issues message AMD071I, and sets the appropriate module return code.

AMDSARMK
XSDSPC

AMDSAXSM - Stand-Alone Dump Extended Storage Management. STEP 11A

AMDSARMK
XSDCURR
AMDSARMK
XSDDER

A. If the PGIN condition code is 1, updates the "data error" count and issues message AMD071I.

MSG071: 13
EFRM071, EFRMID, DERR071

AMDSARMK
XSDDER

AMDSARMK
XSDCURR
AMDSARMK
XSDBNAC

B. If the PGIN condition code is 3, updates the "block not available" count and issues message AMD071I.

MSG071: 13
EFRM071, EFRMID, BNA071

AMDSARMK
XSDBNAC

ESTE
ESTVLSID ESTCHNG ESTDATA
XPTE
XPTDATA
XPTE
XPTLSID
AMDSARMK
RCXSMAUX

C. If a valid copy of the page exists on auxiliary storage, puts the LSID of the page into the BCT and sets the module return code to 4 to indicate an error occurred on the PGIN instruction, but a valid copy of the page exists on auxiliary storage.

AMDSARMK
BCTLSID

AMDSARMK
RCXSMERR

D. If no valid copy of the page exists on auxiliary storage, sets the module return code to 8 to indicate that an error occurred on the PGIN instruction and no valid copy of the page exists on auxiliary storage.

12 **Returns to the subroutine caller.**

AMDSAXSM - Stand-Alone Dump Extended Storage Management. **STEP 13**

PARAMETERS

INSERT1 INSERT2
INSERT3

13

MSG071

13 Issues message AMD071I.

PARAMETERS

INSERT1

14 Sets up the inserts for the message. Converts the E-FRAME number or the ESTE pointer to EBCDIC.

SVCI
SVCBIN, INSERT2, INS2071, LENGTH(INSERT2)

PARAMETERS

INSERT3

15 Writes the message.

IO
MSGID(AMD071I)

16 Returns to the subroutine caller.

Section 3: Program Organization

This section contains the following information about the organization of the stand-alone dump program:

- A discussion of the three stages of processing.
  - The specification stage
  - The initialization stage
  - The execution stage
- A series of figures illustrating the various storage layouts during different stages of processing.
- A description of module calling sequences.
- Module and macro descriptions.

Three Stages of Processing

Specification Stage

During the specification stage, you set up SADMP the way you want it. You specify the type of SADMP program to run, and specify whether additional storage is to be dumped. You code the AMDSADMP macro instruction, specifying keyword values that tailor the dump program to the installation. Then you assemble the macro instruction using either of two methods:

- In two-stage generation, you first assemble the AMDSADMP macro instruction to produce stage-two JCL. Then, you execute this JCL to initialize the SADMP residence volume.
- In one-step generation, after coding the AMDSADMP macro as input control statements, you execute the AMDSAOSG program as a single job step.

Initialization Stage

The initialization of the SADMP residence volume involves three phases:

1. Assembly of the AMDSADM2 macro produces the SADMP real storage dump program AMDSARDM, which dumps real storage, and the SADMP common communication table (AMDSACCT), an internal control block.
2. The SADMP build module AMDSABLD puts the output from phase one onto the residence volume in ready-to-load form. AMDSABLD locates the SADMP IPL program (AMDSAIPL) and the SADMP virtual storage dump program (AMDSAPGE), and puts them onto the residence volume.
3. If the residence volume is on a direct access device, the device utility ICKDSF is invoked to put SADMP's IPL text onto the device's IPL track (cylinder 0, track 0).

4. If the residence volume is on a tape, that tape must be initialized with a tape mark and have the file protect ring inserted.

Execution Stage

The execution of SADMP involves three processes:

1. Channel programs IPL1 and IPL2 load AMDSAIPL into page 0 of real storage. Then the IPL process gives control to AMDSAIPL. AMDSAIPL loads AMDSACCT into storage, then loads and calls AMDSARDM for the real storage dump program and AMDSAPGE for the virtual storage dump program. AMDSADIP locates storage that does not contain data needed by systems, and loads AMDSAPGE into this storage.

2. AMDSARDM dumps real storage and processor-related information, and writes title and header records.

3. AMDSAPGE dumps paged-out virtual storage.

Notes:

1. *Stand-alone dump uses only real, online devices. When dumping to or from devices that have both real and virtual storage, specify only real storage addresses to SADMP. Stand-alone dump must reside in real, online storage.*

2. *You cannot direct SADMP output to its residence volume.*

The SADMP IPL process destroys valuable data because it must include a processor reset. Therefore, the first part of SADMP is the processor STORE STATUS (See *MVS/XA Principles of Operation*), done prior to IPL. This copies the volatile data into main storage at locations in the PSA which, because of their hardware usage, are not likely to be of value to MVS.

The IPL reads 24 bytes of data (IPL1) into location 0, then executes the channel program beginning at location 8. When the program successfully completes, the IPL sequence completes by putting the IPL device subsystem ID into X'30' and loading the PSW at location 0.

The IPL1 channel program reads in up to X'90' bytes of data, loads the IPL2 channel program into location X'110', and issues a transfer in channel instruction (TIC) to it. The data that IPL1 and IPL2 overlaid is lost; fortunately, it is of no value to SADMP or for debugging. IPL2 saves the remainder of absolute page 0 by writing it onto the IPL device, and reads module AMDSAIPL into X'840' bytes of storage beginning at storage location X'7C0'.

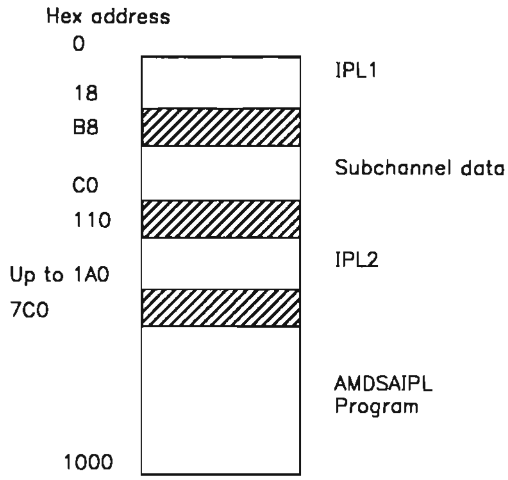


Figure 4-3. SADMP Absolute Storage after IPL.

When the channel program completes, the PSW that is loaded gives control to AMDSAIPL. AMDSAIPL copies additional storage onto the residence device that provides additional working storage. AMDSAIPL loads AMDSARDM and calls AMDSARDM to dump the processor data and real storage, substituting the data saved for use by the virtual dump, which requires the data areas to reconstruct the virtual storage. To accomplish this, the data overlaid when AMDSARDM was loaded is copied back in. (See Figure 4-4).

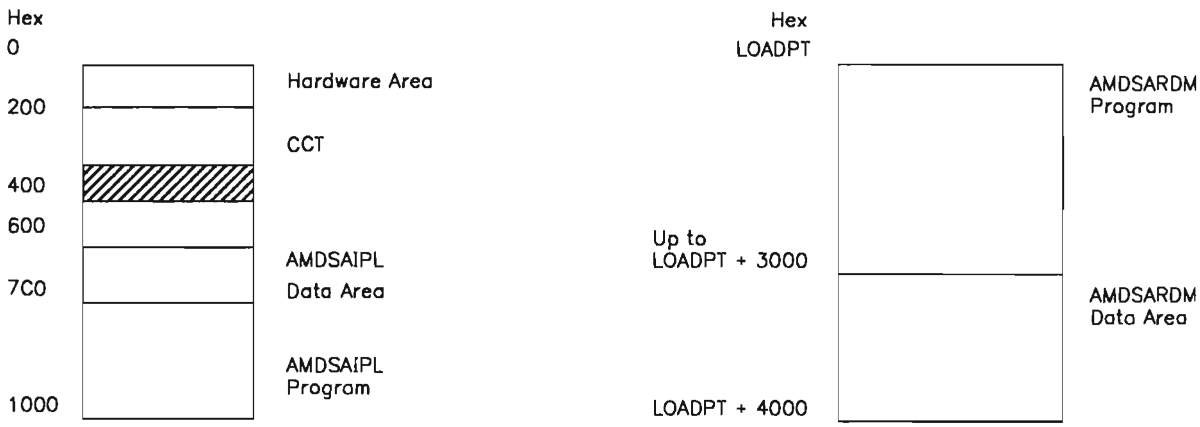


Figure 4-4. SADMP Absolute Storage During AMDSARDM Execution.

AMDSAIPL then initiates the loading of AMDSAPGE which dumps the paged-out virtual storage. During the real storage dump, a PSA is located and information is copied into the CCT. The 4k bytes that it occupied contains no data required by the virtual dump. It can be used as working storage. AMDSAIPL reads the first 4K bytes of AMDSAPGE (CSECT AMDSADIP) into this page, and gives it control.

AMDSADIP locates storage management control blocks and uses them to determine real storage that is either backing the link pack area or available (backing nothing). AMDSADIP builds the storage use table SUT to describe this storage. AMDSADIP also locates a range of contiguous virtual addresses in the

LPA, which it remaps (by altering the page table entries) onto the real storage found earlier. The data contained in that range of virtual addresses cannot be dumped. The content of the LPA is sufficiently constant that the operator DUMP command can obtain the contents at a later time.

Once AMDSADIP obtains the storage, AMDSADIP loads the remaining pages of AMDSAPGE using an I/O subroutine in AMDSAIPL. Following AMDSADIP in the load module is the relocation table (RLT) put there by AMDSABLD during residence volume initialization (see Figure 4-5, Figure 4-6, and Figure 4-7). AMDSADIP uses the RLT to relocate the address constants, both real and virtual, in the rest of AMDSAPGE. After loading is complete, AMDSAIPL passes control to CSECT AMDSAPGE for initialization.

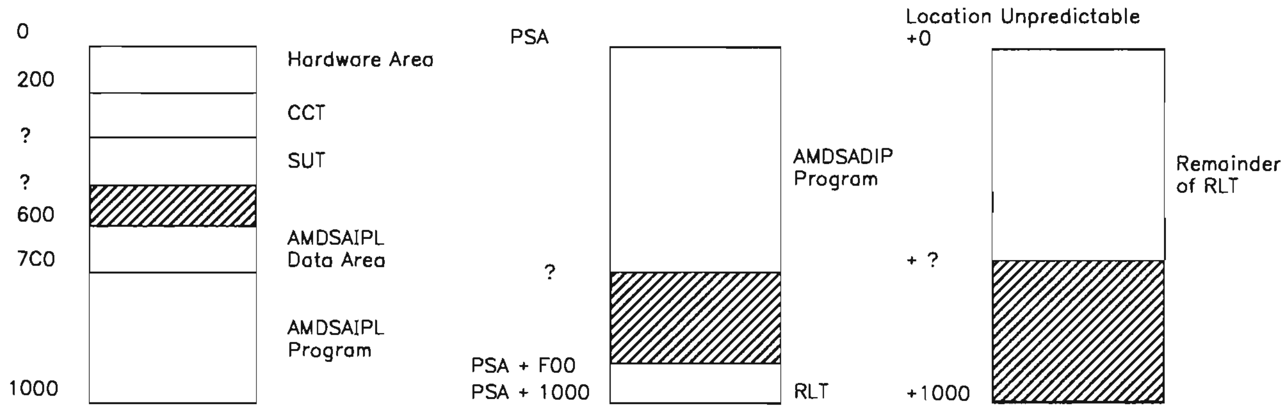


Figure 4-5. SADMP Absolute Storage During AMDSADIP Execution. AMDSADIP and the RLT are in Contiguous Virtual Storage.

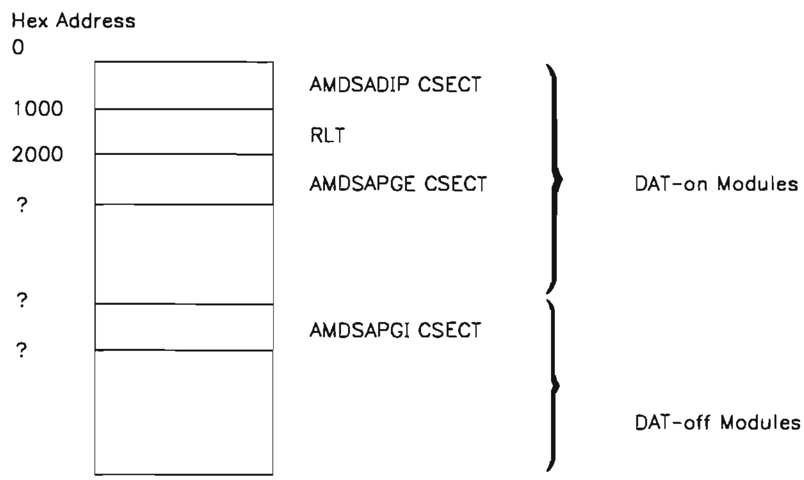


Figure 4-6. AMDSAPGE Load Module

On the residence volume, this is divided into 4K blocks.

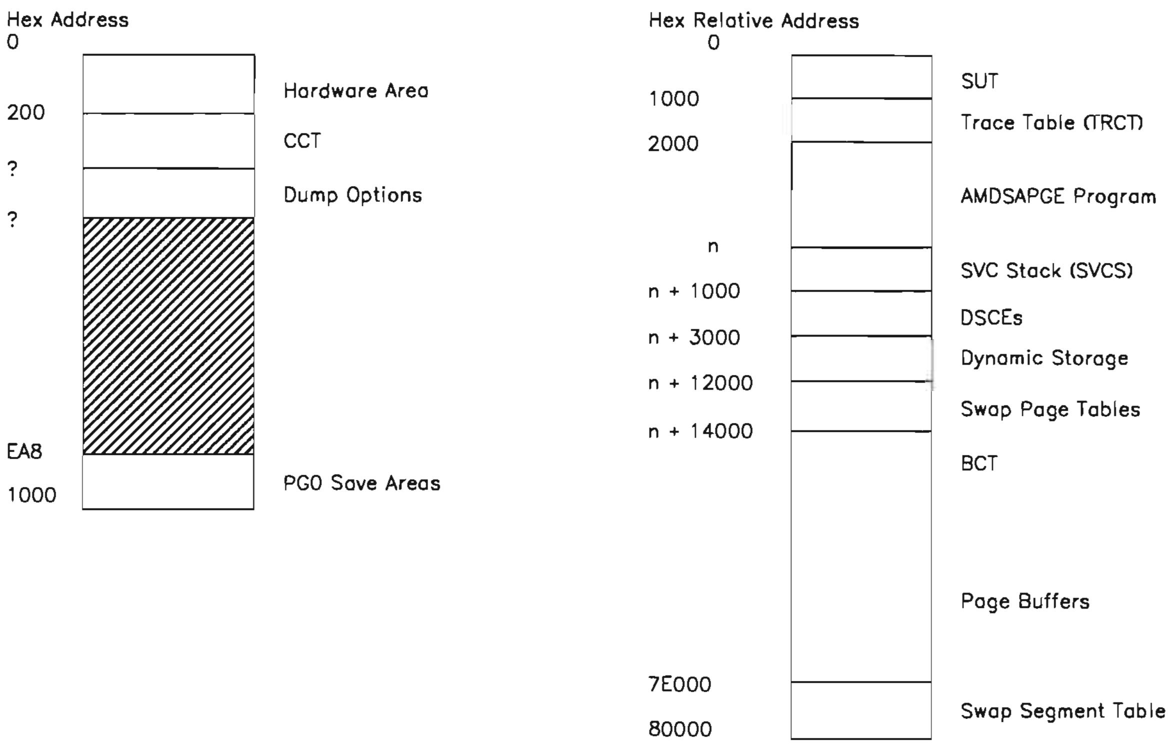


Figure 4-7. SADMP Virtual Storage During AMDSAPGE Execution

The SADMP page table may be at n - 1000. The SUT address is at CTSUT(virtual). (See "Section 4: Data Areas" in this chapter.)

FILE #	REC#	Hex Length	Content
1	1	18	IPL1
	2	Up to 90	IPL2
	3	840	Module AMDSAIPL
	4	1K	AMDSACCT communications table
	5	12K	Real storage dump module AMDSARDM
	6 to n	4K	Virtual storage dump load AMDSAPGE
2	1	4K	Storage save area for 0-4K
	2, 3, 4	4K	Storage save area for the 12K used by AMDSARDM

Figure 4-8. Tape Residence Volume Format

DDNAME	REC #	Hex Length	Content
IPLDEV	1	4K	AMDSACCT communications table
	2, 3, 4, 5	4K	storage save area for the 16K user by AMDSARDM
	6, 7, 8, 9	4K	real storage dump module AMDSARDM
	10 to N	4K	virtual storage dump load module AMDSAPGE
TRKOTEXT	1	24	IPL1
	2	90	IPL2
	3	4K	storage save area for 0-4K
	4	840	module AMDSAIPL

Figure 4-9. DASD Residence Volume Format

It is not possible to write over a particular record on a tape. To avoid loss of data, all writing must take place following the last record on the tape that is to be preserved. Refer to Figure 4-8 and Figure 4-9 to determine the location, length and description of specific records when using tape or DASD residence volumes.

The virtual storage dump program is a single machine-executable module (AMDSAPGE) that consists of a number of object modules. These modules provide operating system functions that are necessary to access data from different address spaces. For example, AMDSAPGE contains a program-interrupt handler, auxiliary- and real-storage managers, getmain and freemain routines, an I/O interrupt handler, I/O error-recovery routines, and an SVC interrupt handler.

Linkage between modules in AMDSAPGE is through the use of SVCs; an AMDSADMP SVC interrupt handler performs the linkage. Each module is assigned an SVC number which other modules use when calling the module. The SVC calling scheme provides for dynamic address translation for modules that must operate as part of each different address space processed. See Section 2: Method of Operation, for more information.

Module Calling Sequences for AMDSADMP Functions

The following maps show sequential flow of AMDSADMP modules. Each map indicates the active modules for a specific function and describes the operations performed by those modules. Entry point names are also provided where they differ from the module names. Figure 4-10 explains the design of sequence maps.

For more detailed descriptions of the functions, see the diagrams in Section 2: Method of Operation.

Single Path Flow

```
Calling Module
|
├─ Module A
|  |
|  ├─ Module A1
|  |
|  └─ Module A2
|
└─ Module B (ENTRY1)
```

The calling module first calls A, which then calls A^1 and A^2. When A returns, the caller passes control to B at entry point ENTRY1.

Alternate Path Flow

```
Calling Module
|
└─┬─ Module A
  |
  └─ Module B
     |
     └─ Module B1
```

The calling module determines whether to call A or B for this operation. If B receives control, it calls B^1 before returning to the caller.

Figure 4-10. Example of Calling Sequence Map

AMDSADMP Generation

One-Step Generation

- AMDSAOSG — Initializes the residence volume in one job step.
  - AMDSADMP — Accepts macro specification of SADMP and invokes AMDSADM2.
    - AMDSADM2 — Generates the real dump program AMDSARDM and the CCT.
  - AMDSABLD — Initializes the SADMP program in SYS1.PAGEDUMP.
    - AMDSAPRS — Checks the syntax of the dump options.
  - ICKDSF — Puts SADMP IPL text onto track 0, cylinder 0 of a direct access device.

Two-Stage Generation

- AMDSADMP — Accepts macro specification of SADMP and produces stage-two JCL.
  - AMDSADM2 — Generates the real dump program AMDSARDM and the CCT.
  - AMDSABLD — Initializes the SADMP program in SYS1.PAGEDUMP.
  - ICKDSF — Puts SADMP IPL text onto track 0, cylinder 0 of a direct access device.

AMDSADMP Execution

- AMDSABLD (IPL1 channel program) — IPL channel program, which reads in IPL2.
  - AMDSABLD (IPL2 channel program) — Saves page 0 on IPL volume and reads in AMDSAIPL.
    - AMDSAIPL — SADMP IPL program. Reads the CCT, AMDSARDM and AMDSADIP into real storage and gives them control.
      - AMDSADM2 (AMDSARDM) — Writes header and processor records and dumps real storage.
      - AMDSADIP — Loads the virtual storage dump program into real and virtual storage.
        - AMDSAPGE — Dumps paged-out virtual storage and console trace data.

Console Message Formatting

AMDSAFCM — Formats the SADMP console message log.

Virtual Storage Dump Program

- AMDSAPGE — Initializes the virtual storage dump program and controls dumping.
  - AMDSAPGE(AMDSADMP) — Dumps console trace data and paged-out virtual storage.
    - AMDSAMSF — Dumps console trace data.
    - AMDSAVCK — Checks the validity of the ASM and address space control blocks.
    - AMDSAPMT — Asks the operator for additional storage to dump.
    - AMDSAAID — Dumps local address spaces.
    - AMDSASIO — Rewinds and unloads the output tape.
    - AMDSACON — Informs the operator that the dump is done.
    - AMDSADCM — Initializes and dumps the console message log.

Dump of Console Trace Data

- AMDSAMSF — Dumps console trace data.
  - AMDSABUF — Obtains a page buffer.
  - AMDSASIO — Writes the page buffer to the output tape.

Dump of Address Spaces

- AMDSAAID — Locates and initializes address spaces, and causes dumping of data in them.
  - AMDSAVCK — Checks the validity of the ASCB.
  - AMDSASIN — Initializes (swaps in) the address space.
  - AMDSAMDM — Dumps relevant data.
  - AMDSAGTF — Dumps GTF trace data from the GTF address space.
  - AMDSACON — Issues address-space-statistics messages.

Dump of Address Space Data

AMDSAMDM — Scans the dump table and dumps the data in the address space that the dump table describes.

- AMDSATXT (AMDSABIN) — Converts binary data to EBCDIC.
- AMDSACON — Writes messages to the console.
- AMDSAGTM — Obtains and frees storage for the VSMLIST work area.
- IGVSLIST — Converts high-level descriptions of storage (subpools, LSQA, etc.) into address ranges.
- AMDSARRD — Dumps address ranges.
  - AMDSAAUD — Logs errors and recovers from them.
- AMDSAAUD — Logs errors and recovers from them.
- AMDSAVCK — Checks the validity of address space control blocks.

Note: IGVSLIST (VSMLIST) is a program that resides in the nucleus of the dumped system and provides the SADMP interface to virtual storage management.

Dump of GTF Data

AMDSAGTF — Dumps the GTF history queue.

- AMDSAVCK — Checks the validity of the GTF control blocks.
- AMDSATXT (AMDSABIN) — Converts binary fields to EBCDIC.
- AMDSACON — Writes messages to the console.

Prompting for Additional Storage

AMDSAPMT — Processes dump options specified when SADMP was generated, and prompts the operator to specify additional dump options. Adds additional storage to the dump table.

- AMDSAGTM — Allocates and frees storage for AMDSAPMT buffers and data.
- AMDSACON — Writes messages to the console.
- AMDSAUUD — Logs errors after program checks, GETMAIN failures, or other unusual events.
- AMDSAPRS — Checks the dump option syntax and translates dump option input text into a dump option set.
  - AMDSAGTM — Allocates and frees storage.

Note: AMDSAPRS must run under both SADMP and MVS/XA. AMDSAPRS issues SVC 120 for storage, which under SADMP becomes a call to AMDSAGTM.

Page/Segment Fault Handling and Dumping

- AMDSAUPD — Resolves page and segment faults, and dumps all pages brought in as a result.
  - AMDSAGTM — Obtains and frees the autodata area.
  - AMDSABUF — Allocates a page buffer.
  - AMDSARSM — Locates the faulting page in real or auxiliary storage.
  - AMDSADOS — Copies real pages.
  - AMDSAASM — Converts an LSID into a TTR and device address.
  - AMDSASIO — Reads the page from auxiliary storage and dumps it to the output tape.
  - AMDSAAUD — Passes control to the recovery environment if the fault cannot be resolved.

Console Message Dump

- AMDSADCM — Writes console messages to the output tape.
  - AMDSAGTM — Obtains and frees the autodata area.
  - AMDSAAUD — Recovers from unexpected errors.
  - AMDSABUF — Obtains dump page buffers.
  - AMDSASIO — Writes message-dump buffers to the output tape.

Real Storage Management Interface

- AMDSADIP — Determines which page frames the virtual dump can be safely loaded into.
- AMDSASIN — Initializes (swaps in) address spaces.
  - AMDSAVCK — Checks the validity of control blocks
  - AMDSABUF — Obtains page buffers.
  - AMDSADOS — Copies real pages.
  - AMDSAAUD — Recovers from errors and logs them.
  - AMDSASIO — Dumps page buffers.
  - AMDSACON — Writes console messages.
  - AMDSATXT (AMDSABIN) — Converts data to EBCDIC.
  - AMDSAASM — Translates LSIDs into a device address and TTR.
  - AMDSAXSM — Reads pages from extended storage.
- AMDSARSM — Locates a faulting page in real or auxiliary storage.
  - AMDSAGTM — Obtains and frees autodata area.
  - AMDSAXSM — Reads pages from extended storage.

Auxiliary Storage Management Interface IOS Interface

AMDSAASM	— Converts a logical slot identifier into a direct access device seek/search address and device address.
├── AMDSAGTM	— Obtains and frees autodata area.
├── AMDSAVCK	— Checks the validity of ASM control blocks.
└── AMDSAUCB	— Converts a UCB address into a SADMP IODB.
├── AMDSAGTM	— Obtains and frees autodata area.
├── AMDSAVCK	— Checks the validity of the UCB.
├── AMDSATXT (AMDSABIN)	— Converts binary data to EBCDIC.
├── AMDSACON	— Writes messages to the console.
├── AMDSAAUD	— Logs errors.
└── AMDSAUCB	— Locates other exposures of a multiple-exposure device.

Extended Storage Interface

AMDSAXSM	— Extended storage manager.
├── AMDSAGTM	— Obtains and frees autodata storage.
├── AMDSATXT(AMDSABIN)	— Converts binary data to printable EBCDIC.
└── AMDSACON	— Writes console messages.

Control Block Validity Checking

AMDSAVCK	— Checks the validity of control blocks.
├── AMDSAGTM	— Obtains and frees autodata storage.
├── AMDSAAUD	— Recovers from errors.
├── AMDSATXT (AMDSABIN)	— Converts binary data to EBCDIC.
└── AMDSACON	— Writes messages to the console.

Operator-Controlled Termination and Restart

Recovery (AMDSAEXI)	— Handles external interruptions as requests for premature termination of SADMP.
├── AMDSACON	— Writes messages to the console.
└── AMDSAAUD	— Terminates SADMP.
AMDSAPGE (AMDSASRR)	— Reinitializes and restarts the virtual storage dump program.
├── AMDSAUUD	— Takes a diagnostic dump.
└── AMDSASIO	— Waits for all I/O to complete.

Page Buffer Allocation

AMDSABUF	— Allocates, reclaims and steals page buffers.
├── AMDSAGTM	— Obtains and frees autodata storage.
└── AMDSASIO	— Waits for tape I/O to complete, and free buffers.

I/O Device Error Recovery

AMDSADER	— Recovers from errors on direct access devices.
├── AMDSAGTM	— Obtains and frees autodata storage.
├── AMDSASIO	— Executes DASD channel programs.
├── AMDSACON	— Writes messages to the console.
├── AMDSATXT (AMDSABIN)	— Converts binary data to EBCDIC.
├── AMDSATER (AMDSAERB)	— Updates error recovery block.
├── AMDSATER (AMDSAERM)	— Writes I/O error messages.
├── AMDSATER (AMDSAINR)	— Requests intervention on as direct access device.
└── AMDSASIO (AMDSATCP)	— Translates real channel program addresses to virtual addresses.
AMDSATER	— Recovers from tape I/O errors and handles end-of-tape processing.
├── AMDSAGTM	— Obtains and frees autodata storage.
├── AMDSASIO	— Executes tape channel programs.
├── AMDSACON	— Writes messages to the console.
├── AMDSATXT (AMDSABIN)	— Converts binary data to EBCDIC.
├── AMDSATER (AMDSAERB)	— Updates error recovery block.
├── AMDSATER (AMDSAERM)	— Writes I/O error messages.
├── AMDSATER (AMDSAINR)	— Requests intervention on a tape.
├── AMDSATER (AMDSANTP)	— Unloads old tape and causes a new one to be mounted.
├── AMDSAAUD	— Terminates SADMP if the tape is unusable.
└── AMDSASIO (AMDSATCP)	— Translates real channel program addresses to virtual addresses.
AMDSATER (AMDSAERB)	— Updates device error recovery blocks.
└── AMDSAGTM	— Obtains and frees autodata storage.
AMDSATER (AMDSAERM)	— Writes I/O error messages.
├── AMDSAGTM	— Obtains and frees autodata storage.
├── AMDSATXT (AMDSABIN)	— Converts binary data to EBCDIC.
└── AMDSACON	— Writes messages to the console.

- AMDSATER (AMDSAINR) — Requests intervention on direct access devices.
  - AMDSAGTM — Obtains and frees autodata storage.
  - AMDSACON — Writes messages to the console.
- AMDSASIO — Waits for intervention.
- AMDSATER (AMDSANTP) — Unloads old tape and causes a new one to be mounted.
  - AMDSAGTM — Obtains and frees auto data storage.
  - AMDSASIO — Executes tape channel programs to write tape marks, rewind, unload, and read labels.
  - AMDSACON — Writes messages to the console.
  - AMDSATER (AMDSAINR) — Requests intervention on a tape.
- AMDSAT80 — Recovers from tape I/O errors for 3480 tape drives.
  - AMDSAGTM — Obtains and frees auto data storage.
  - AMDSASIO — Executes tape channel programs.
  - AMDSACON — Writes messages to the console.
  - AMDSATER (AMDSAERB) — Updates device error recovery block.
  - AMDSATER (AMDSAERM) — Writes I/O error messages.
  - AMDSATER (AMDSAINR) — Requests intervention on a tape.
  - AMDSATER (AMDSANTP) — Unloads old tape and causes a new one to be mounted.
  - AMDSATXT (AMDSABIN) — Converts binary data to EBCDIC.
  - AMDSAAUD — Terminates SADMP if the tape is unusable.
  - AMDSASIO (AMDSATCP) — Translates real channel program addresses to virtual addresses.

I/O Services

- AMDSASIO — Performs I/O operations by providing channel programs and issuing SSCH.
  - AMDSAGTM — Obtains and frees autodata area.
  - AMDSATXT (AMDSABIN) — Converts binary data to EBCDIC.
  - AMDSACON — Writes messages to the console.
  - AMDSASIO — Waits for I/O to complete.
  - AMDSAAUD — Terminates SADMP after fatal errors.
- AMDSAIOI — Handles I/O interruption (called via on I/O interruption).
  - AMDSASIO (AMDSATCP) — Translates virtual channel program addresses to real addresses.

AMDSAIOI — Handles I/O interruption, recovers from errors and frees used BCT.

- AMDSAGTM — Obtains and frees autodata area.
- AMDSASIO — Restarts channel programs.
- AMDSATER — Recovers from tape errors.
- AMDSAT80 — Recovers from tape errors for 3480 tape drive.
- AMDSADER — Recovers from DASD errors.

AMDSACON — Writes messages to the console and reads replies.

- AMDSASIO — Executes console channel programs.
- AMDSAAUD — Recovers from errors.

Basic Services

AMDSAPGI — Handles program interruptions.

- AMDSAAUD — Recovers from errors, takes internal dumps, and abnormally terminates SADMP.
- AMDSAUPD — Resolves page faults.

AMDSASIO (AMDSATCP) — Translates channel program addresses from real addresses to virtual addresses and from virtual addresses to real addresses.

- AMDSAGTM — Obtains and frees autodata storage.
- AMDSAAUD — Passes control to the top RCB on the error recovery stack if the channel program is too long.

AMDSASVI — Handles SVC interruptions and sets up execution environments.

- AMDSAAUD — Recovers from errors.

AMDSATXT (AMDSABIN) — Converts binary data to EBCDIC.

AMDSATXT (AMDSAEBC) — Converts EBCDIC data to binary.

AMDSAGTM — Allocates and frees dynamic storage.

- AMDSAAUD — Recovers from errors.

AMDSAAUD — Recovers from errors, logs errors, takes dumps, and abnormally terminates SADMP.

- AMDSAGTM (AMDSAFRM) — Frees storage owned by abnormally terminating processes.
- AMDSADCM — Dumps console message dump buffers.
- AMDSACON — Writes a message to the operator console.
- AMDSASIO — Rewinds and unloads the tape and dumps SADMP storage to the output tape.

Module/Macro Descriptions

This list includes all CSECTs, entry points, included segments, and executable macros.

Module/Macro	Description
AMDSAAID	Searches MVS/XA for all defined address spaces.
AMDSAARD	Accumulates a list of address ranges dumped for an address space.
AMDSAASM	Translates an LSID into a page data set CCHHR.
AMDSAAUD	Implements an FRR-like recovery mechanism and manages the self-dumping and abnormal termination functions.
AMDSABIN	Alternate entry to AMDSATXT, that converts binary to printable EBCDIC.
AMDSABLD	Problem program that organizes and prepares the SADMP code for easy loading, and creates the IPL channel program.
AMDSABUF	Manages the BCT (buffers/page frames) allocation, reclamation, and stealing.
AMDSACCW	Contains the tape and DASD channel programs.
AMDSACON	Provides the console communication service and contains the console CCWs.
AMDSADCM	Dumps console messages to the output tape.
AMDSADER	DASD error recovery procedures (ERP).
AMDSADIP	Locates storage (real and virtual) for the virtual dump code, then loads and relocates it.
AMDSADMP	Macro that checks the validity of the generation parameters, and punches either the job stream that initializes the residence volume or, the input to AMDSAOSG.
AMDSADM2	Macro that generates the various real storage dump routines.
AMDSADOS	Performs DAT-off operations.
AMDSAEBC	Alternate entry to AMDSATXT, that converts printable EBCDIC to binary.
AMDSAERB	Alternte entry to AMDSATER, that updates the ERB.
AMDSAERM	Alternate entry to AMDSATER, that produces ERP error messages.
AMDSAEXI	Alternate entry AMDSAPGE, that handles external interruptions.
AMDSAFCM	Formats the console message log.
AMDSAFRM	Alternate entry AMDSAGTM that AMDSAAUD uses to recover short short term storage allocated by modules that are being terminated.
AMDSAGTF	Dumps the GTF trace data.
AMDSAGTM	Manages the dynamic storage by providing support for the R-form GETMAIN macro (SVC 10 and 120).
AMDSAINR	Alternate entry to AMDSATER, which handles required intervention.
AMDSAIOI	Handles I/O interruptions.
AMDSAIPL	Bootstraps all of SADMP into the processor with minimal data loss.
AMDSAMDM	Manages the dumping of an address space.

AMDSAMSG	Problem program message CSECT.
AMDSANTP	Unloads a tape and causes a new one to be mounted.
AMDSAOSG	A problem program that uses dynamic allocation and invocation to perform the residence volume initialization as a single job.
AMDSAPGE	Completes the virtual dump initialization after AMDSADIP loads the code.
AMDSAPGI	Program interruption handler.
AMDSAPMT	Handles interactive prompting for additional storage to be dumped.
AMDSAPRS	Parses the responses to interactive prompting.
AMDSARDM	Module generated by macro AMDSADM2 to dump real storage.
AMDSARMK	Included segment that contains the executable macros for internal interfaces, the data area mappings, and constants.
AMDSARSM	Determines the frame address or LSID of a page for AMDSAUPD.
AMDSASAD	Contains a TYPE=SAD expansion of AMDSADM2 that supports AMDSAAUD by dumping real storage and communicating with the console.
AMDSASA2	Contains the second half of AMDSASAD to allow division into two noncontiguous pages.
AMDSASAM	Alternate entry into AMDSASAD to write messages to the console.
AMDSASAR	Alternate entry into AMDSASAD to unload the output tape.
AMDSASIN	Swaps in an address space by establishing the minimum DAT structure.
AMDSASIO	Drives all I/O and manages the I/O for all devices.
AMDSASSR	Alternate entry into AMDSAPGE that performs a system restart by reinitializing the virtual dump.
AMDSASVI	Provides an SVC linkage service that allows address space switching, DAT mode switching, flow tracing, and recovery check pointing.
AMDSATCP	Translates channel program addresses from real addresses to virtual addresses or from virtual addresses to real addresses.
AMDSATER	Tape error recovery procedures (ERP).
AMDSATXT	Message CSECT for AMDSACON.
AMDSAT80	Tape error recovery procedures (ERP) for 3480 tape drives.
AMDSAUCB	Converts UCB address into a SADMP IODB.
AMDSAUPD	Manages the page/segment fault handling.
AMDSAVCK	Checks the validity of control blocks and data areas.
AMDSAXSM	Reads pages from extended storage.
SGAMA401	SYSGEN macro for link-edited SADMP code.
SGAMA501	SYSGEN macro for copied SADMP code.

Macro Interfaces

Module/Macro	Description
?ERROR	Indicates an unusual or error condition and the action to be taken.
?FREEMAIN	Dynamic storage deallocation.
?GETMAIN	Dynamic storage allocation.
?IO	I/O services: DASD, tape, console, and UCB to IODB conversion.
?LVA	Converts a real address to a virtual address within SADMP.
?RCBREGS	Makes sure that registers are not used across an RCB exit.
?RECOVER	Initializes, enqueues, and dequeues an RCB.
?SVCI	Creates an internal PLS procedure that facilitates use of a PLS call with a parameter list for service routines that have no specific macro interface.
?TRACE	Places entries into the trace table.
?TRANSCHP	Translates channel program addresses.
?UPDERB	Keeps track of I/O retries.
?VCHK	Checks the validity of control blocks.

Section 4: Data Areas

Buffer Control Table (BCT)

Size: 72 bytes.

Created by: AMDSAPGE.

Updated by: AMDSABUF, AMDSADER, AMDSAIOI, AMDSRSM, AMDSASIN, AMDSASIO, AMDSATER, AMDSADCM, AMDSAUPD.

Pointed to by: CTBCTAVQ (available queue head), BCTAVQP (available queue chain), CTBCTALL (all BCT queue head), BCTQUEUE (all BCT queue head), IODBIOQP (output I/O queue head), BCTIOQP (output I/O queue chain). All pointers are real addresses.

Use: Represents a SADMP page buffer.

BCTs are initialized on the available queue. Whenever a page of data needs to be brought in from real or virtual storage or needs to be dumped, a BCT is allocated for a paging operation. After the operation completes, the BCT is placed on the I/O queue to be written to the dump output device. When output I/O completes, the BCT is returned to the available queue unless it contains information (such as a page table) important for further execution.

If no BCT is available to satisfy a request, a BCT is stolen according to an LRU algorithm. BCTs associated with a terminated address space are automatically reclaimed.

BCT

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	64	BCT	Buffer control table entry.
0	(0)	CHARACTER	4	BCTBCT	EBCDIC acronym.
4	(4)	CHARACTER	16	BCTINIT	This field must be cleared when a BCT is made available.
4	(4)	ADDRESS	4	BCTPGTEP	Address of page table entry that points to the page in the buffer. For non-master address spaces, the page table is in a SADMP storage buffer and this address is common virtual. For the master address space the page table may also be in private storage. In either case, the PGTE is accessible from master.
8	(8)	ADDRESS	4	BCTSGTEP	Common virtual or master virtual address of the segment table entry that points to the page table in the buffer.
12	(C)	BITSTRING	1	BCTUSE	Buffer usage. If zero, BCT is available.
		1...		BCTINUSE	This buffer contains a page which is in use for DAT.
		.1..		BCTPGING	This BCT is involved in a swapping/paging operation.
		.1		BCTIO	The I/O service is processing this BCT.
		...1		BCTPGT	The buffer contains a page table.
		 xxxx			Reserved.

Offsets		Type	Length	Name	Description
13	(D)	BITSTRING	1	BCTSTAT	Buffer status flags.
		x...	....		Reserved.
		.1..	....		BCTRECLM Virtual address reclaimed from real storage.
		..1.	....		BCTSWPDS Page from swap data set.
14	(E)	BITSTRING	2	BCTPICOD	Address-fault program interrupt code.
16	(10)	UNSIGNED	4	BCTUSID	Unreferenced steal interval count down. Initialize to X'FFFFFFFF' if reference bit for buffer is on when AMDSABUF steals a buffer; otherwise, it is decreased by one
20	(14)	CHARACTER	8		Queue pointers(virtual addresses).
20	(14)	ADDRESS	4	BCTQUEUE	Chain field for all BCTs.
24	(18)	ADDRESS	4	BCTIOQP	Next BCT on I/O queue.
24	(18)	ADDRESS	4	BCTAVQP	Next BCT on available queue.
28	(1C)	ADDRESS	4	BCTFRAME	Real address of frame to be reclaimed.
28	(1C)	UNSIGNED	4	BCTLSID	Logical slot identifier; page's auxilary storage id.
28	(1C)	UNSIGNED	1		Unused.
29	(1D)	UNSIGNED	1	BCTINDEX	PART/SART entry number.
30	(1E)	UNSIGNED	2	BCTRBA	Relative block address.
32	(20)	ADDRESS	4	BCTBUFRA	Real address of buffer.
36	(24)	CHARACTER	8	BCTRCHD	Header portion of output record containing the page in the buffer.
36	(24)	ADDRESS	1	BCTKEY1	Storage key of first 2K.
37	(25)	ADDRESS	1	BCTKEY2	Storage key of last 2K.
38	(26)	UNSIGNED	2	BCTASID	ASID of data in buffer.
40	(28)	ADDRESS	4	BCTVADDR	Virtual address of data in buffer (original ASID).
40	(28)	CHARACTER	2		Reserved.
42	(2A)	CHARACTER	2	BCTCPU	CPU address of the console loop trace from which the trace was taken.
44	(2C)	ADDRESS	4	BCTBUFVA	Virtual address of buffer in SADMP virtual storage.
48	(30)	CHARACTER	16	BCTPTRD	Page table reclaim data. This field must be cleared when a BCT is made available.
48	(30)	ADDRESS	4	BCTPTBCT	Common virtual address of the BCT for the page table that maps the page in this BCT's buffer. If 0, the page table is not in a BCT buffer. This pointer is used to update BCTPTPGS in the page table's BCT.
52	(34)	SIGNED	2	BCTPTGS	The number of pages mapped by the page table in this buffer. If 0, none are mapped, or this page does not contain a page table, and this BCT may be safely reclaimed. If not 0, this BCT may not be reclaimed.
54	(36)	CHARACTER	10		Reserved.

OCP

Size: 24 bytes.

Created by: AMDSASIO.

Updated by: AMDSASIO.

Pointed to by: CPB.

Use: OCP is a channel program you can use to write a dump record header and BCT page buffer to the output tape.

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	24	OCP	Writes dump record header and BCT page buffer to an output tape.
0	(0)	CHARACTER	8	OCPCCW1	CCW to write page header.
0	(0)	CHARACTER	1	OCP1CMD	Write.

Offsets		Type	Length	Name	Description
1	(1)	CHARACTER	1	OCP1FLGS	CD, SLI.
		1111			
		.... 1...		OCP1PCI	Bit is on to cause an I/O interrupt after the previous interrupt ends.
2	(2)	ADDRESS	2	OCP1LEN	Length of BCTRCHD.
4	(4)	ADDRESS	4	OCP1ADDR	Address of BCTRCHD.
8	(8)	CHARACTER	8	OCPCCW2	CCW writes page in buffer.
8	(8)	CHARACTER	1	OCP2CMD	Not used.
9	(9)	CHARACTER	1	OCP2FLGS	CC, SLI.
10	(A)	ADDRESS	2	OCP2LEN	Length of buffer (4K).
12	(C)	ADDRESS	4	OCP2ADDR	Address of buffer.
16	(10)	CHARACTER	8	OCPCCW3	TIC to channel program, next BCT or a NOP.
16	(10)	CHARACTER	1	OCP3CMD	NOP or TIC.
17	(11)	CHARACTER	1	OCP3FLGS	Must be 0 for TIC.
18	(12)	ADDRESS	2	OCP3LEN	Must be 0 for TIC.
20	(14)	ADDRESS	4	OCP3ADDR	Unused or address of next OCP.

Common Communication/Control Table (CCT)

Size: 1024 byte record on the residence volume, 64 bytes during AMDSARDM execution, 140 bytes during virtual storage dump program execution.

Created by: AMDSADM2 macro. AMDSAIPL, AMDSARDM, and AMDSAPGE complete the functional initialization.

Updated by: All SADMP programs that run at execution time.

Located at: Absolute 512 = X'200'

Use: Central control block for SADMP. The CCT provides communication among the IPL program, the real storage dump program, and the virtual storage dump program. The CCT also provides either pointers to most SADMP shared data areas, or the data areas themselves.

Offset		Size	Field Name	Description
0	(0)	80	CTINITED	This section of the CCT is initialized in the form of the CSECT AMDSACCT, which is generated at residence volume initialization time.
0	(0)	8		EBCDIC ID.
8	(8)	4	CTLOADPT	AMDSARDM load point (page boundary).
12	(C)	4	CTLOADEN	First byte after the storage used by AMDSARDM.
16	(10)	4	CTIPLSID	IPL device's subchannel ID.
20	(14)	4	CTCCHHS	Beginning of SYS1.PAGEDUMP.
24	(18)	4	CTCCHHE	Ending CCHH of SYS1.PAGEDUMP.
28	(1C)	1		Unused.
29	(1D)	1	CTINRPT	Number of record tracks for IPL device which contains SYS1.PAGEDUMP.
30	(1E)	2	CTINTPC	Number of tracks/cylinders for IPL device.
32	(20)	4	CTOUTSID	Output device subchannel ID.
36	(24)	4	CTCONSID	Console subchannel ID.
40	(28)	4	CTLINES	Number of lines current on the console screen. This is used to control screen erasure and to communicate between AMDSARDM and AMDSACON.
44	(2C)	1	CTCONTYP	Console type.
		1...	CTCT3277	3277/3278 type.

Offset		Size		Field Name	Description
45	(2D)	1			CTOPTION These flags represent generation options.
		1...		....	CTIPLTYP OFF=DASD, ON=TAPE IPL device type.
		.1..		....	CTDMPFMT Output format ON=PRINTED, OFF=MACHINE readable.
		..1.		....	CTVIRTD For printed dumps only, the default addresses to be used in taking the dump if the operator cannot communicate with AMDSARDM. OFF=REAL, ON=VIRTUAL.
		...1		....	CTMSG OFF=ALL, ON=ACTION only messages are to be written to the console.
		....		1...	CTPROMPT OFF=DO NOT, ON=DO prompt for dump tailoring options.
46	(2E)	1			CTFLGRS Flags reset on retry
		1...		....	CTSCREND OFF=PROMPT before erase; ON=ERASE then continue.
		.1..		....	CTNODUMP Page buffers must not be written to the output tape.
		..1.		....	CTCONACT Messages should not be issued during error recovery.
47	(2F)	1			CTFLGNR Flags not reset on retry.
		x...		....	Reserved.
		.1..		....	CTNOCONS No functioning console exists.
48	(30)	4		CTGENDTO	Address of the dump tailoring option input from the DUMP=keyword at generation time.
52	(34)	4		CTPREFIX	Processor prefix value obtained from store status by AMDSARDM. This is the real address where AMDSADIP is loaded, and the SUT resides.
56	(38)	4		CTCVTP	CVT virtual address.
60	(3C)	4		CTSTDM	Segment table designation of master address space.
64	(40)	4		CTPCCAV	PCCA virtual address.
68	(44)	4		CTSUT	Virtual address of the storage use table.
72	(48)	4		CTRCB	RCB queue head (virtual).
76	(4C)	4		CTXSD	Extended storage descriptor address.
80	(50)	60		CTUNINIT	This portion of the CCT is used only by the virtual dump modules, and is not initialized in AMDSACCT. It is set to zeros by AMDSAPGE during its initialization.
80	(50)	4		CTIODB	Head of the queue of all IODB (virtual).
84	(54)	4		CTSVCSTV	SVC status stack pointer (virtual).
88	(58)	4		CTTRACE	Trace table address (real).
92	(5C)	4		CTBCTAVQ	Queue of available BCT.
96	(60)	4		CTBCTALL	Queue of all BCT.
100	(64)	4		CTAUDPL	Parameter list for AMDSAUUD (?ERROR).
100	(64)	1		CTAUDACT	Error recovery activity flags.
		1...	....	CTAUDRCR	No recursion permitted through AMDSAUUD.
		.1..	....	CTAUDIOX	Error recovery I/O cannot be called.
		..1.	....	CTAUDPCE	A program check is expected.
101	(65)	1		CTAUDFLG	Flags.
		x...	....		Unused.
		.1..	....	CTAUDDMP	Dump requested.
		..1.	....	CTAUDTRM	Termination requested.
		...1	....	CTAUDRCV	Recovery requested.
102	(66)	1		CTAUDRC	Reason code.
103	(67)	1		CTAUDSVC	SVC number.
104	(68)	20			Not used.
104	(68)	2		CTASIDA	Active ASID.
106	(6A)	2		CTASIDD	ASID being dumped.
108	(6C)	4		CTASCBM	Master address space ASCB address.
112	(70)	4		CTASCBA	Active address space ASCB address.
116	(74)	4		CTASCBD	ASCB address for address space being dumped.
120	(78)	4		CTSTDD	STD for address space being dumped. STD for address space being dumped. Used by AMDSASVI for routine that run in address space being dumped.

Dump Table

Size: 26 bytes minimum, 4096 bytes maximum.

Created by: AMDSAPGE.

Updated by: AMDSAPMT.

Pointed to by: CTDT (virtual).

Use: Describes the paged-out storage that the virtual dump program must dump in addition to common storage, GTF data and console trace data.

Offset		Size	Field Name	Description
0	(0)	8	DTHEADER	The dump table header.
0	(0)	4	DTHID	EBCDIC acronym 'DTAB'.
4	(4)	2	DTHALLOC	Length of the storage allocated for this table (including header).
6	(6)	2	DTHLEN	Length of table (including header) in bytes.
8	(8)	0	DTENTRY	Table entries.
0	(0)	2	DTLIST	Generic name for any/all of the parts of a dump table entry (DTENTRY).
0	(0)	2	DTCOUNT	The number of members in the array for the list being mapped.
2	(2)	0	DTARRAY	The array of list members. Each PART entry is defined in this field.
2	(2)	0	DTAS	Address space identifier (ASID) list.
2	(2)	0	DTASINT	ASID interval array.
2	(2)	2	DTASLO	Smallest ASID in interval.
4	(4)	2	DTASHI	Largest ASID in interval.
2	(2)	0	DTJN	Job name list.
2	(2)	0	DTJOBNAM	Job name array.
2	(2)	0	DTKEY	Storage key list.
2	(2)	0	DTKEYINT	Key interval array.
2	(2)	1	DTKEYLO	Smallest key in interval.
3	(3)	1	DTKEYHI	Largest key in interval.
2	(2)	0	DTADDR	Address range list.
2	(2)	0	DTADDRIN	Address interval array.
2	(2)	4	DTADDRLO	Smallest address interval.
6	(6)	4	DTADDRHI	Largest address in interval.
2	(2)	0	DTSP	Subpool list.
2	(2)	0	DTSPINT	Subpool interval array.
2	(2)	1	DTSPLO	Smallest subpool in interval.
3	(3)	1	DTSPHI	Largest subpool in interval.

Dynamic Storage Control Element (DSCE)

Size: 16 bytes each: 256 DSCEs make up a 4096 byte DSCE area.

Created by: AMDSAPGE.

Updated by: AMDSAGTM, AMDSAFRM.

Pointed to by: SUTDSCEU (in-use queue head), SUTDSCEA (available queue head), DSCESHRT (available queue and short-term storage queue chain), DSCELONG (long-term storage queue chain).

Use: Describes a block of storage that is either for dynamic allocation, or already allocated. An available for describing a new block of storage. An in-use DSCE describes unallocated storage, or describes storage that is allocated for short-term or long term use. An in-use DSCE for short-term storage is automatically freed if the process that requested the DSCE terminates abnormally. An in-use DSCE that is allocated for long term storage is never deallocated unless deallocation is explicitly requested. In-use DSCEs are circularly double-threaded by the long and short chain fields.

Offset		Size	Field Name	Description
0	(0)	4	DSCESHRT	Queue chain used to satisfy short term requests.
0	(0)	4	DSCEAQ	The SUTDSCEA queue.
4	(4)	4	DSCELONG	Queue chain used to satisfy long term requests.
8	(8)	4	DSCEADDR	Real address of the beginning of the area described.
12	(C)	2	DSCELEN	Length of the area.
14	(E)	1	DSCESPN	Subpool number. 128 or higher is for long-term request. 15 (F) 1 DSCESSI SVC stack index that was current when this storage was allocated. Zero if storage is unallocated.

Input/Output Device Block (IODB)

Size: 92 bytes.

Created by:. AMDSAUCB (paging device IODBs), AMDSAPGE (all others).

Updated by: All virtual dump modules that use I/O.

Pointed to by: CTIODB (IODB queue head), IODBNEXT (IODB queue chain).

Use: Represents an I/O device, the current status of the device, and any I/O operation taking place on the device.

IODB

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	88	IODB	Input/Output Device Block.
0	(0)	CHARACTER	32	IODBQCA	Queuing control area. QCARIGHT places all the IODB onto a singly threaded queue. QCADOWN points to a CPB queue of channel programs to be executed.
0	(0)	CHARACTER	4	IODBIODB	IODB acronym.
32	(20)	CHARACTER	16	IODBDSCH	Description of I/O device and subchannel.
32	(20)	CHARACTER	8	IODBCDID	Device type and model numbers for the control unit and device, in packed decimal, as received from the sense ID command (e.g.FF38801133500100 from a 3350 model 1 with a 3800 model 11 control unit.
32	(20)	CHARACTER	1		Reserved. Must be X'FF'.
33	(21)	CHARACTER	3	IODBCUID	Control unit ID.
33	(21)	CHARACTER	2	IODBCUT	Control unit type number.
35	(23)	CHARACTER	1	IODBCUM	Control unit model number.
36	(24)	CHARACTER	3	IODBDID	Device ID.
36	(24)	CHARACTER	2	IODBDT	Device type number.
38	(26)	CHARACTER	1	IODBDM	Device model number.
39	(27)	CHARACTER	1		Reserved.
40	(28)	UNSIGNED	2	IODBDEV	Device number.
42	(2A)	UNSIGNED	2	IODBSCH	Subchannel ID.

Offsets		Type		Length	Name	Description
44	(2C)	BITSTRING		1	IODBCLAS	Device class copied from an MVS UCB or equal to the value defined for UCBs.
45	(2D)	BITSTRING		1	IODBUTYP	Unit type copied from an MVS UCB or equal to the value defined for UCBs.
46	(2E)	BITSTRING		1	IODBISCM	Interrupt subclass mask for control register 6.
47	(2F)	BITSTRING		1	IODBUSE	Use to which the device is put.
		1...	....		IODBOUT	Dump output volume.
		.1..	....		IODBPAGE	Page dataset.
		..1.	....		IODBCONS	Operator console.
		...1	....		IODBIPL	SADMP IPL device (dump residence volume).
48	(30)	CHARACTER		16	IODBOPER	Describes the I/O operation. The standard parameter list for AMDSASIO is the IODB followed by 4 parameters that are copied in order into the 4 unreserved subfields of IODBOPER.
48	(30)	BITSTRING		4	IODBOPTY	Operation type flags.
		x...	....			Reserved.
		.1..	....		IODBCLOS	Close a device.
		..1.	....		IODBREAD	Read from a device.
		...1	....		IODBWRIT	Write to a device.
		....	1...		IODBSNSC	Sense.
		....	.x..			Reserved.
		....	..1.		IODBCHP	The caller has provided a channel program address (IODBCHPA).
		....	...1		IODBWAIT	IODBWAIT=1, IODBFINT=0, start any specific I/O operation, then if an interrupt is expected wait until no more interrupts are expected. IODBWAIT=1, IODBFINT=1 same, but return after the first interrupt.
49	(31)	1...	....		IODBRCVR	Attempt to recover from I/O errors by calling the device ERP module.
		.1..	....		IODBINRQ	Intervention request. Wait for an interrupt even if the device is not active.
		..1.	....		IODBCHPR	The channel program virtually addressed by IODBCCWA has real, not virtual address fields (used when the I/O service is restarting a channel program).
		...1	....		IODBFINT	Return after the first interrupt.
		....	1111			
50	(32)	1111	1111			
51	(33)	xxxx	xxxx			Reserved.
52	(34)	ADDRESS		4	IODBCHPA	Address of a channel program provided by the I/O service user.
56	(38)	ADDRESS		4	IODBREQ	Address of SADMP control block that represents the buffer for the requested I/O operation.
60	(3C)	CHARACTER		1		Reserved.
61	(3D)	BITSTRING		3	IODBWRC	If nonzero, this value is to be used as the wait code while waiting for this I/O.
64	(40)	CHARACTER		20	IODBSTAT	Status of the device and subchannel and their current I/O operation.
64	(40)	UNSIGNED		2	IODBWCNT	Number of processes waiting for I/O.
66	(42)	BITSTRING		2	IODBSFLG	Status flags.
		1...	....		IODBUNAV	Device unusable.
		.1..	....		IODBINTX	Interrupt is expected I/O is currently active.
		..xx	xx..			Reserved.
		....	..1.		IODBERR	The last I/O interrupt presented an I/O error that has not yet been corrected.
		....	...1		IODBENE	Error in sense command.

Offsets		Type	Length	Name	Description
67	(43)	1...	....	IODBDERP	Device error recovery procedures are in progress.
		.xxx	xxxx		Reserved.
68	(44)	ADDRESS	4	IODBSCHB	Address of SCHIB. Its SCSW is updated with the most recent SCSW from a TSCH.
72	(48)	ADDRESS	4	IODBCCWA	CCW address suitable for restarting the current channel program. This is either the last CCW started by SSCH or the last nonzero CCW address from an SCSW.
76	(4C)	ADDRESS	4	IODBSENS	Address of sense data.
80	(50)	UNSIGNED	1	IODBSENL	Length of sense data.
81	(51)	CHARACTER	3	IODBERB	Error Recovery Block.
81	(51)	CHARACTER	3	ERB	
81	(51)	CHARACTER	1	ERBINSID	Message ID.
82	(52)	UNSIGNED	1	ERBRCNT	Retry count.
83	(53)	UNSIGNED	1	ERBMAX	Maximum number of retries left.
84	(54)	ADDRESS	4	IODBDDX	Address of device dependent extension.

DDX

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	8	DDX	Device dependent IODB extension, common fields.
0	(0)	CHARACTER	4	DDXID	ID field.
4	(4)	ADDRESS	4	DDXDDX	Pointer to a secondary DDX.

DDXD — direct access extension. Some field names begin with IOB so they will agree with pre-JBB2125 usage.

0	(0)	STRUCTURE	24	DDXD	Direct access extension.
0	(0)	CHARACTER	8		ID field 'DDXD' and secondary extension pointer (DDXI).
8	(8)	ADDRESS	4	IODBUCB	Virtual address of the UCB for this device.
12	(C)	CHARACTER	8	IODBRID	For a page-in operation, this describes where in the page dataset to find the record that contains the page. Includes the seek and search arguments.
12	(C)	CHARACTER	8	IODBCCHH	Seek address.
12	(C)	CHARACTER	1	IODBM	'M' value.
13	(D)	CHARACTER	2	IODBSEEK	Beginning of seek address.
15	(F)	CHARACTER	4	IODBSRCH	Search address.
15	(F)	CHARACTER	2	IODBCC	Cylinder number.
17	(11)	CHARACTER	2	IODBHH	Track number.
19	(13)	CHARACTER	1	IODBR	Record number.
20	(14)	BITSTRING	1	IODBFL6	Copied from UCBFL6.
21	(15)	BITSTRING	1	IODBDFLG	DASD flags.
		1...		IODBPGST	This device is a cached auxiliary storage subsystem (see DDXI).
		.x..			Reserved.
		..1.		IODBNG	The UCB for this device is bad.
22	(16)	CHARACTER	2		Reserved.

DDXI - secondary extension for cached auxiliary storage subsystem (CASS) (e.g., 3880/11 + 3350). These devices are DASDs with a cache and microcode optimized for paging. They use different CCWs and access UCBs differently from ordinary DASD.

DDXC - console extension.

0	(0)	STRUCTURE	9	DDXC	Console extension.
0	(0)	CHARACTER	8		ID is 'DDXC'.
8	(8)	BITSTRING	1	DDXCFLAG	Flags.
		x...			Reserved.
		.1..		DDXCFWAT	Wait at the end of the next full screen and ask for permission to wait at the following screen.

DDXT - tape extension.

0	(0)	STRUCTURE	8	DDXT	Tape extension.
0	(0)	CHARACTER	8		ID is 'DDXT'.

QCA

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	32	QCA	Queuing control area.
0	(0)	CHARACTER	8	QCAELEMT	Description of the element this QCA is in.
0	(0)	CHARACTER	4	QCAETYP1	EDCDIC ID to identify the type of the element.
4	(4)	CHARACTER	2		Reserved.
6	(6)	UNSIGNED	2	QCAELEN	Element length, including the QCA.
8	(8)	CHARACTER	18	QCASTRUC	Description of the structure this QCA is implementing.
8	(8)	ADDRESS	4	QCALEFT	Left pointer for trees and doubly threaded queues.
12	(C)	ADDRESS	4	QCARIGHT	Right pointer for trees and doubly threaded queues, also pointer for singly threaded structures and stacks.
16	(10)	ADDRESS	4	QCAUP	Pointer to owning or higher level entity. This may be a unit of processing the control block is waiting for.
20	(14)	ADDRESS	4	QCADOWN	Pointer to owned or lower level entity.
24	(18)	BITSTRING	1	QCASTYP1	Type of data structure.
25	(19)	BITSTRING	1	QCASTYP2	Subtype of data structure.
26	(1A)	CHARACTER	6		Reserved.

DCA

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	64	DCA	DAT change communication area at X'600'.
0	(0)	CHARACTER	4	DCAID	EBCDIC ID 'DCA'.
4	(4)	BITSTRING	4	DCAFUNC	Function control bits.
		1...		DCAFCOPY	1 = copy.
		.1..		DCAFSRCA	1 = source is described by address in 2nd parm.
		..1.		DCAFLEN	1 = 3rd parm is length of data.
8	(8)	CHARACTER	56	DCAPARMS	Parameters.
8	(8)	UNSIGNED	4	DCAPARM1	1st parameter.
8	(8)	ADDRESS	4	DCAPRCVR	Receiver of operation.
12	(C)	UNSIGNED	4	DCAPARM2	2nd parameter.
12	(C)	ADDRESS	4	DCAPSRCE	Source of operation.
16	(10)	UNSIGNED	4	DCAPARM3	3rd parameter.
16	(10)	UNSIGNED	4	DCAPLEN	Length of data.
20	(14)	CHARACTER	44		Reserved.

TMD

Offsets		Type		Length	Name	Description
0	(0)	STRUCTURE		17	TMD	Tape Message Display.
0	(0)	CHARACTER		1	TMDFCNTL	Format control byte.sq..
		111.	....		TMDFOVLY	New message overlay 000 = General 001 = Dismount 010 = Mount 111 = Dismount/Mount.
		...1	....		TMDFALT	Alternating message 0 = Display 1 msg only 1 = Alternate messages.
		....	1...		TMDFBLNK	Blinking message 0 = Do not blink msg 1 = Blink message.
		....	.1..		TMDFDSPL	Display message 0 = Display 1st msg only = Display 2nd msg only ignored if TMDFALT = 1.
		....	..xx			Reserved.
1	(1)	CHARACTER		8	TMDMSG1	1st Display Message.
1	(1)	CHARACTER		1	TMDM1ACT	Operator Action.
2	(2)	CHARACTER		6	TMDMIVSR	Tape Volume Serial Number.
2	(2)	CHARACTER		5	TMDM1VNM	Constant 'SADMP' text.
7	(7)	CHARACTER		1	TMDM1VN8	Variable Number (1 to 9).
8	(8)	CHARACTER		1	TMDM1AMD	Action Modifier.
9	(9)	CHARACTER		8	TMDMSG2	2nd Display Message.
9	(9)	CHARACTER		1	TMDM2ACT	Operator Action.
10	(A)	CHARACTER		6	TMDM2VSR	Tape Volume Serial Number.
10	(A)	CHARACTER		5	TMDM2VNM	Constant 'SADMP' text.
15	(F)	CHARACTER		1	TMDM2VN8	Variable Number (1 to 9).
16	(10)	CHARACTER		1	TMDM2AMD	Action Modifier.

Message Parameter (MSGPARM)

Size: 12 bytes.

Created by: AMDSADM2, AMDSATXT.

Updated by: AMDSACON, AMDSAPMT.

Use: Describes a write/read sequence to the console I/O service. MSGPARM is located in arrays in AMDSARDM (real dump) and AMDSATXT (virtual dump).

Offset		Size		Field Name	Description
0	(0)	1		MPWRTLEN	Length of text to be written.
1	(1)	1		MPRDLEN	Length of reply to be read.
2	(2)	2		MPRC	Name used to access the 12 bit reason code following MPFLAGS. It must be ANDed with X'0FFF'.
		1111	....	MPFLAGS	
				MPFCONT	When on, indicates that MPLNCNT contains the maximum number of lines which may follow.
		1...	....		This implies that there is no reply possible after this line.
		.1..	....	MPFSPRES	This message might be suppressed.
		..1.	....	MPFVLR	When off, the entire read buffer is folded to uppercase (X'00' to blank). When on, only the character read in is folded.
		...1	....	MPFNCON	Message should go only to the console message dump.
		....	xxxx	Not used.	
		xxxx	xxxx		Reason code used when waiting for the reply. This value is obtained by ANDing MPRC with X'0FFFF'.

Offset		Size	Field Name	Description
4	(4)	4		MPTEXTA Real address of text to be written.
8	(8)	4		PRPYA Real address of reply area.
8	(8)	4		MPLNCNT Maximum number of lines that may follow this one. Only valid if MPFCONT is on.

Page 0 Register Save and PSW Build Areas (PG0MAP)

Size: 344 bytes.

Created by: AMDSAPGE.

Updated by: AMDSAAUD, AMDSAIOI, AMDSAPGI, AMDSASVI.

Located at: Absolute 3752 (X'EA8'). The end of PG0MAP is always at the end of page 0.

Use: Register save areas and PSW build areas for interrupt handlers and other programs that cannot rely on base registers to locate data areas. PG0MAP also contains common SVC instructions.

Offset		Size	Field Name	Description
0	(0)	8	PG0PG0	EBCDIC identifier.
8	(8)	64	PG0ACRSA	AMDSAAUD control register save area.
8	(8)	4		
12	(C)	4	PG0ASTD	Control register 1 (segment table designation).
72	(48)	8	PG0APSW	AMDSAAUD PSW to give control to RCB exit.
80	(50)	64	PG0SVISA	Register save area.
80	(50)	4		Register 0.
84	(54)	4	PG0SR1	Register 1.
88	(58)	44		Registers 2-12.
132	(84)	4	PG0SR13	Register 13.
136	(88)	4	PG0SR14	Register 14.
140	(8C)	4	PG0SR15	Register 15.
144	(90)	64	PG0PGISA	Register save area.
208	(D0)	64	PG0IOISA	Register save area.
272	(110)	64	PG0AUDSA	Register save area.
336	(150)	8		Reserved.
336	(150)	2	PG0SVCSR	The restart new PSW points here. This allows AMDSASAVI to prepare the environment.
338	(152)	2	PG0SVCEX	The external new PSW points here. This allows AMDASASVI to prepare the environment.
340	(154)	2	PG0SVCRT	AMDSASVI alters the general purpose register 14 value to point to this as return linkage.
342	(156)	2		Reserved.

Real Storage Management Address Space Data (RAD)

Size: 32 bytes.

Created by: AMDSASIN.

Updated by: AMDSASIN.

Pointed to by: CTRAD (virtual).

Function: Contains information specific to an address space that SADMP needs to resolve translation exceptions. RAD includes a mapping of the virtual storage that the page tables occupy.

Offset		Size	Field Name	Description
0	(0)	4	RADID	Acronym.
4	(4)	4	RADPBEGA	Beginning of above private PGT's.
8	(8)	4	RADPENDA	End of (beyond) above private PGT's.
12	(C)	4	RADPBEGB	Beginning of below private PGT's.
16	(10)	4	RADPENDB	End of below private PGT's.
20	(14)	4	RADPBEGC	Beginning of COMMON PGT's.
24	(18)	4	RADPENDC	End of COMMON PGT's.
28	(1C)	2	RADFHSI	Index of the first segment that contains a private page table. SADMP assumes that the page tables that map this and all segments with larger index are precisely the high page tables (those that map other private page tables).
30	(1E)	2	RADRECNO	Translation exception recursion number. SADMP permits only finite number of recursive page/segment faults.

Recovery Control Block (RCB)

Size: 32 bytes.

Created by: Modules that use the SADMP recovery service and execute DAT-on.

Updated by: AMDSAAUD, and modules that use the SADMP recovery service.

Pointed to by: CTRCB (queue head), RCBNEXT (chain).

Use: Saves an environment (error exit, registers, etc.) that a program needs to regain control when an unusual condition causes the program to terminate its normal sequence of execution.

Offset		Size	Field Name	Description
0	(0)	4	RCBNEXT	Next higher RCB (virtual).
4	(4)	4	RCBSSI	SSINDEX value which can be used to purge calls, because this RCB was queued from the SVC stack. Points beyond the top of the stack when RCB is created.
8	(8)	4	RCBSTD	Control register 1 for exit.
12	(C)	4	RCBSAVE	Address of recovery routine register save area.
16	(10)	4	RCBEXIT	Address of recovery routine. 20 (14) 4 RCBDATA Base address of the dynamic data area for the recovery routine. 24 (18) 4 RCBBASE Base address of recovery routine.

Offset		Size	Field Name	Description
28	(1C)	4		
28	(1C)	1	RCBSM	System mask.
29	(1D)	3	RCBECODE	Error code from AMDSAAUD indicating why this recovery routine was given control.

Relocation Table (RLT)

Size: 4352 bytes (1086 entries) maximum. Each entry is 4 bytes.

Created by: AMDSABLD.

Located at: 3840 bytes (X'F00') into the AMDSAPGE load module on the residence volume before IPL.

Use: Describes the relocatable address constants in the AMDSAPGE load module.

Offset		Size		Field Name	Description
0	(0)	8		RLTHEADR	Table header.
0	(0)	4		RLTRLT	'RLT' acronym.
4	(4)	2		RLTNPGS	Number of pages in AMDSAPGE load module.
6	(6)	2		RLTLEN	Number of records in RLTDATA.
8	(8)	4344		RLTDATA	
Relocation Table Proper					
8	(8)	1		RLTGLAGS	
		x...	....		Reserved.
		.1..	....	RLTDATON	Indicates whether constant pointed to by RLTA is to be interpreted with DAT-ON.
		..xx	xxxx		Reserved.
9	(9)	3		RLTA	Displacement of address constant to be relocated into AMDSAPGE load module.

Sense Data Mapping for I/O Devices

Size: 24 bytes.

Created by: AMDSAPGE, AMDSAUCB.

Updated by: Channel hardware.

Use: Describes the data furnished by a SENSE command. Sense data mapping for I/O devices is located in the device IODB. Symbols used to describe the I/O devices are:

- D = DASD only
- S = 3480 tape only
- T = 3420 tape only
- I = Buffered DASD only

SENSE

Offset		Type		Length	Name	Description
0	(0)	STRUCTURE		32	SENSE	Sense Input Area Mapping.
0	(0)	BITSTRING		1	SENSE00	Sense Byte 0.
		1...	....		SENCR	Command Reject.
		.1..	....		SENIR	Intervention Required.
		..1.	....		SENBO	Bus Out Check.
		...1	....		SENEC	Equipment Check.
		....	1...		SENDC	Data Check.
		....	.1..		SENDA	Deferred Access.
		....	..1.		SENTCC	Track Condition Check (D).
		....	..1.		SENUCT	Unit Check Timing (S).
		....	...1		SENSC	SEEK Check (D).
		....	...1		SENCC	Data Converter Check (T).
		....	...1		SENAE	Assigned Elsewhere (S).
1	(1)	BITSTRING		1	SENSE01	First SENSE byte.
		1...	....		SENPE	Permanent error (D).
		1...	....		SENLBFF	Locate block function failed (S).
		.1..	....		SENITF	Invalid track format (D).
		.1..	....		SENDOCU	Drive online to control unit (S).
		..1.	....		SENEOC	End of cylinder (D).
		..1.	....		SENTB	TU status B bit (T).
		...1	....		SENMTO	Message to operator.
		...1	....		SENOSR	Out-of-sequence record (S).
		....	1...		SENNRF	No record found (D).
		....	1...		SENLP	Load point (T).
		....	1...		SENBOT	Beginning of tape (S).
		....	.1..		SENFPD	File protected DASD (D).
		....	.1..		SENWSTAT	Write status (S).
		....	..1.		SENFPT	File protected tape (TS).
		....	..1.		SENWI	Write inhibited DASD (D).
		....	...1		SENOI	Operation incomplete (D).
		....	...1		SENNC	Tape unit not capable (TS).
2	(2)	BITSTRING		1	SENSEO2	Sense byte 2.
		111.	....		SENSCI	Selecting channel interface (S).
		1...	....		SENBLF	Buffered log full (D).
		.1..	....		SENCO	Correctable (D).
		..1.	....		SENACS	Alternate controller selected (I).
		...1	....		SENEDP	Environmental data present (D).
		...1	....		SENCACU	Channel-adapter control unit (S).
		....	1...		SENMDFCU	Microprocessor data flow control Unit (S).
		....	.xx.			Reserved.
		....	...1		SENBIDPI	Block positioning indicator (S).
3	(3)	CHARACTER		1	SENSEO3	Restart Command code (D).
3	(3)	BITSTRING		1	SENERPAC	Error recovery procedure action Code (S).
4	(4)	CHARACTER		1	SENSEO4	Sense byte 4.
		xxxx	....			Reserved.
		....	1111		SENBIDHI	High order bits; logical block position of block identifier (S).
5	(5)	UNSIGNED		1	SENSEO5	Restart cylinder (D).
		111.	....			Unused.
		...1	....		SENPEID	PE ID burst check (T).
6	(6)	UNSIGNED		1	SENSEO6	Restart track (D).
7	(7)	BITSTRING		1	SENSE07	Format and message.
7	(7)	BITSTRING		1	SENSFMT	Sense data format (S).
		1111	....		SENFMT	Format identifier (I).
		....	1111		SENMSG	Message identifier (I).
8	(8)	CHARACTER		7		Reserved.
15	(F)	CHARACTER		3	SENS1517	Restart displacement (D).
18	(12)	CHARACTER		2	SENS1819	Data error offset (D).

Offset		Type	Length	Name	Description
20	(14)	CHARACTER	4	SENS2023	Correcting data bits 4 bytes ECC (D).
20	(14)	CHARACTER	3	SENS2022	Correcting data Bits 3 byte ECC (D).
24	(18)	CHARACTER	7	SENS2430	Unused by SADMP (S).
31	(1F)	CHARACTER	1	SENS31	Sense byte 31 (S).
31	(1F)	UNSIGNED	1	SENDBCNT	Data byte count (S).

Storage Use Table (SUT)

Size: 540 bytes (129 entries). Each entry is 4 bytes long.

Created by: AMDSADIP.

Updated by: AMDSADIP, AMDSAPGE.

Pointed to by: CTSUT (virtual). CTSUTLOC contains copy of the SUT that AMDSADIP uses. CTSUTLOC does not contain all usage information, but contains a description of the real storage that SADMP uses.

Use: Describes the real and virtual addresses and internal usage of the storage that SADMP occupies. SUT also contains the headers of the DSCE queues and a real pointer to the page frame table.

Offset		Size		Field Name	Description
0	(0)	4		SUTSUT	'SUT' acronym.
4	(4)	4		SUTVIRTB	Beginning of the virtual address range used by AMDSADMP. Maps to SUTFRAMA (0).
8	(8)	2		SUTNFRMS	Index of the last page frame in SUTFRAMA. Note the last two entries in SUTFRAMA will be for contiguous frames.
10	(A)	2		SUTFUNIN	Index of the first frame not assigned a permanent use during initialization.
12	(C)	4		SUTMPFTV	Virtual address of the system's page frame table.
16	(10)	4		SUTDSCEU	Head of the queue of DSCE in use (virtual address).
20	(14)	4		SUTDSCEA	Head of the queue of DSCE available (virtual address).
24	(18)	516		SUTPFT	List of frames used by S/A dump.
24	(18)	4		SUTFRAMA	An entry.
		1...	....		
		.111	1111	SUTFRNUM	The portion of the real address which is the frame number.
		1111	1111		
		1111	....		
27	(1b)	1		SUTUSE	Flags for frame usage.
		1...	....	SUTSELF	Frame contains SUT.
		.1..	....	SUTTRACE	Trace table.
		..1.	....	SUTCODE	S/A dump executable code. S/A dump page table.
		...1	....	SUTPGT	
		....	1...	SUTDSCEP	S/A dump page of DSCE.
		....	.1..	SUTSVCST	SVC status element stack.
		....	..1.	SUTBUF	S/A dump page buffer.
		....	...1	SUTSWAP	SWAP-IN DAT table.

SVC Stack

Size: 4096 bytes.

Created by: AMDSAPGE.

Updated by: AMDSAAUD, AMDSASVI.

Pointed to by: CTSVCSTV

Use: Represents the current status of SVC calls. An SVC status element is placed onto the stack when an SVC call is made and is removed from the stack when the called module returns.

SVC Status Element

Size: 96 bytes.

Created by: AMDSASVI.

Updated by: AMDSASVI.

Referred to by: The address of the SVC stack (CTSVCSTR real, CTSVCSTV virtual) and the index SSINDEX of the current top entry.

Function: Describes the status of a process that was suspended when it made an SVC call.

Offset		Size		Field Name	Description
0	(0)	8		SSHEAD	Unique header portion.
0	(0)	4		SVCSVC	Acronym.
0	(4)	4		SSINDEX	Index of the entry currently in use.
8	(8)	4032		SSARRAY	The stack.
0	(0)	96		SVCSTAT	SVC status element.
0	(0)	8		SSPSW	SVC old PSW value.
0	(0)	4			
		1111	1...		
		....	.1..	SSPDAT	DAT Mode.
4	(4)	4		SSPIA	Address of instruction following SVC.
8	(8)	4		SSTDCR1	Segment table designation (CR1).
12	(C)	1		SSNUM	Called SVC number.
13	(D)	1			
14	(E)	2		SSASID	CTASIDA value.
16	(10)	4		SSASCB	CTASCBA value.
20	(14)	4		SSORIG14	Register 14 original value because the address of a return SVC is substituted.
24	(18)	72		SSSA	Save area passed to called module address in register 13.
24	(18)	4			
28	(1C)	4		SSAR13	Register 13 original back savearea pointer received from caller.

Swap Address Table (SAT)

Size: 36 bytes.

Created by: AMDSAPGE.

Pointed to by: CTSAT (virtual).

Use: Describes the real addresses and master address space virtual addresses of the DAT tables used to initialize an address space.

Offset		Size	Field Name	Description
0	(0)	4	SATID	Acronym.
4	(4)	4	SATMSV	Virtual address of the master segment table.
8	(8)	4	SATSPV	Virtual address of SADMP's own page table.
12	(C)	4	SATMSV	Virtual address in master of the local memory's segment table.
16	(10)	4	SATLSR	Real address of the 8K contiguous real storage used for local segment tables.
20	(14)	4	SATLPTV	Virtual address in master of the local memory's top page table.
24	(18)	4	SATLPTR	Real address of the frame used for the local memory's top page table, if it wasn't in real storage already.
28	(1C)	4	SATLP0V	Virtual address in master of the local memory's segment 0 page table.
32	(20)	4	SATLP0R	Real address of the frame used for the local memory's 0 page table, it it wasn't in real storage.

Trace Table

Size: 4096 bytes. Each entry is 8 bytes long.

Created by: AMDSAPGE.

Updated by: AMDSAAUD, AMDSAGTM, ADMSAIOI, AMDSAPGI, AMDSASVI.

Pointed to by: CTTRACE (real).

Use: Records the 511 most recent significant events that occur during execution of the virtual storage dump program (SVC calls and returns, program and I/O interrupts, interrupt handler exits, errors, and others).

Offset		Size	Field Name	Description
0	(0)	4	TRCTRC	EBCDIC Identifier.
4	(4)	4	TRCINDEX	Index of the last entry used.
8	(8)	4088	TRCTABLE	Trace table entry. This is basically a PSW having additional information.
8	(8)	3		Front of PSW.
11	(B)	1	TRCID	Identifier of the module getting control. TRCID is usually an SVC number. For interrupt handlers a pseudo SVC number is assigned.
12	(C)	4	TRCMADDR	Mode and address.
12	(C)	4	TRCINTAD	Address of the interrupt.

Section 5: Diagnostic Aids

This section contains:

- A description of AMDSADMP wait reason codes.
- A description of AMDSADMP error codes.
- A description of the trace table.
- A description of the SVC linkage stack.

Note: Refer to *System Messages* for message numbers, message text, and detecting, issuing, and containing modules.

Wait Reason Codes

Wait reason codes identify unusual or error conditions which might occur in SADMP and provide specific diagnostic information when SADMP is not running. Each wait reason code is issued at only one location in the SADMP code even though the same condition might be detected at more than one location. A wait reason code is not intended to identify a condition, but identifies how the condition was detected.

A wait reason code is 3 bytes long:

- The first byte is a wait code that indicates the condition.
- The last two bytes are the reason code.

For a description of SADMP wait reason codes, refer to *Service Aids*.

Error Codes

The low-speed real dump program may be used to diagnose problems in the high-speed real or virtual dump program. Error codes appear in error trace records and terminating wait PSWs. They show abnormal conditions in SADMP ranging in severity from common and not serious to terminating. The rightmost three bytes of the PSW or trace entry are X'4Fssee'. (ss) is the SVC number or trace event ID (pseudo-SVC) of the issuing program, and (ee) is the unique error identifier.

The virtual dump portion of the high-speed dump program takes diagnostic dumps to the output tape following a failure in the virtual dump program. A message is issued to the console informing the operator of the failure and the module in which the failure may have occurred. An ASID associated with the failure is also provided. To aid in determining the cause of the dump, a field in the CCT (CTAUDIT) is provided which may identify the problem. The values and meanings in CTAUDIT are:

Error Code	Explanation	Module
X'0301'	Retry is done for a console I/O error.	AMDSACON
X'0302'	The console is lost due to an I/O error.	AMDSACON
X'0801'	The segment table is invalid.	AMDSASIN

Error Code	Explanation	Module
X'0802'	The top page table is invalid.	AMDSASIN
X'0803'	The page table for segment is invalid.	AMDSASIN
X'0804'	No BCT is available.	AMDSASIN
X'0805'	AMDSAASM failed.	AMDSASIN
X'0806'	The SFT is invalid.	AMDSASIN
X'0807'	A DAT table was not read in.	AMDSASIN
X'0901'	No BCT is available for paging.	AMDSAUPD
X'0904'	RSM control block errors.	AMDSAUPD
X'0905'	AMDSAASM failed.	AMDSAUPD
X'0906'	SADMP is unable to read the page from the data set.	AMDSAUPD
X'0907'	No BCT is available for output.	AMDSAUPD
X'0910'	SADMP is unable to resolve the segment fault.	AMDSAUPD
X'0911'	SADMP is unable to resolve the page fault.	AMDSAUPD
X'0A01'	GETMAIN requested more than 4096 bytes.	AMDSAGTM
X'0A02'	No more DCSEs are available.	AMDSAGTM
X'0A03'	No storage is available.	AMDSAGTM
X'0A04'	FREEMAIN was for unallocated storage.	AMDSAGTM
X'0D01'	Recursion through AMDSAAUD.	AMDSAAUD
X'0D02'	More than 2 self-dumps requested.	AMDSAAUD
X'0D03'	Terminating error in or recursive entry to the I/O service.	AMDSAAUD
X'0D04'	Master segment table designation in CCT is invalid.	AMDSAAUD
X'0D05'	SADMP is unable to recover because RCB queue is empty.	AMDSAAUD
X'1001'	An unexpected error occurred.	AMDSAARD
X'1002'	GETMAIN for ARBs failed.	AMDSAARD
X'1001'	An unexpected error occurred.	AMDSAMDM
X'1102'	GETMAIN FOR VSM work area failed.	AMDSAMDM
X'1103'	ASXB is invalid.	AMDSAMDM
X'1104'	An unexpected error occurred while scanning the dump table.	AMDSAMDM
X'1105'	VSMLIST program check processing LSQA.	AMDSAMDM
X'1106'	VSMLIST program check processing CSA.	AMDSAMDM
X'1107'	VSMLIST program check processing SQA.	AMDSAMDM
X'1108'	VSMLIST program check processing private.	AMDSAMDM
X'1109'	VSMLIST program check processing subpool list.	AMDSAMDM
X'110A'	An error occurred during backward TCB search.	AMDSAMDM
X'110B'	An error occurred during forward TCB search.	AMDSAMDM
X'110C'	End of TCB queue was not reached before the count was exhausted.	AMDSAMDM
X'110D'	AMDSAARD failed.	AMDSAMDM
X'1120'	Nonzero return code from VSMLIST on LSQA request.	AMDSAMDM
X'1121'	Nonzero return code from VSMLIST on CSA request.	AMDSAMDM
X'1122'	Nonzero return code from VSMLIST on SQA request.	AMDSAMDM
X'1123'	Nonzero return code from VSMLIST on private request.	AMDSAMDM
X'1124'	Nonzero return code from VSMLIST on subpool list request.	AMDSAMDM
X'1201'	Recursion limit through AMDSARSM exceeded.	AMDSARSM
X'13F1'	A permanent I/O error exists on the output tape.	AMDSATER
X'1604'	The UCB is invalid.	AMDSAUCB
X'1603'	An invalid subchannel ID exists in UCB.	AMDSAUCB
X'160C'	The subchannel is not operational.	AMDSAUCB
X'160'	The subchannel is invalid.	AMDSAUCB
X'1701'	The input is invalid.	AMDSAVCK
X'1B01'	An unresolved page fault exists in AMDSAPMT or AMDSAPRS.	AMDSAPMT
X'1B02'	Dump table overflow.	AMDSAPMT
X'1B03'	No console.	AMDSAPMT

Error Code	Explanation	Module
X'1C13'	System restart.	AMDSASRR
X'1D12'	An external interrupt terminated dumping.	AMDSAEXI
X'1E01'	An error occurred during I/O initialization.	AMDSAPGE
X'1E02'	An error occurred during dynamic storage management. initialization.	AMDSAPGE
X'1E03'	An error occurred during BCT initialization.	AMDSAPGE
X'1E04'	An error occurred during SVC service initialization.	AMDSAPGE
X'1E05'	An error occurred during external interrupt handler initialization.	AMDSAPGE
X'1E06'	An error occurred during swapping service initialization.	AMDSAPGE
X'1E07'	An error occurred during trace table initialization.	AMDSAPGE
X'1E08'	ASMVT is invalid.	AMDSAPGE
X'1E09'	ASVT is invalid.	AMDSAPGE
X'1E0A'	PART is invalid.	AMDSAPGE
X'1E0B'	SART is invalid.	AMDSAPGE
X'1E0C'	An error occurred during system restart initialization.	AMDSAPGE
X'1E0D'	An error occurred during machine check handler initialization.	AMDSAPGE
X'1E0E'	An error occurred calling through SVC service to AMDSADMP entry point.	AMDSAPGE
X'1E0F'	An error occurred in a dumping routine.	AMDSAPGE
X'1E10'	An error occurred during dump table initialization.	AMDSAPGE
X'1E11'	An error occurred when calling AMDSAPMT.	AMDSAPGE
X'1E13'	An error occurred during system restart.	AMDSAPGE
X'1E14'	Assign failed for a 3480 tape drive.	AMDSAPGE
X'2101'	A channel program for address translation is longer than 1024 bytes.	AMDSATCP
X'22F1'	A terminating I/O error occurred on the output tape.	AMDSAT80
X'7801'	GETMAIN requested more than 4096 bytes.	AMDSAGTM
X'7802'	No DSCE is available.	AMDSAGTM
X'7803'	No storage is available.	AMDSAGTM
X'7804'	FREEMAIN was for unallocated storage.	AMDSAGTM
X'F901'	The SVC stack is full. A loop probably exists in SADMP	AMDSASVI
X'FDF1'	A termination error occurred on the output tape (no paths available).	AMDSAIOI
X'FEcc'	Program check. The last byte of the interrupt code is cc.	AMDSAPGI
X'FEFF'	Recursion through the program interrupt handler.	AMDSAPGI

Trace Table

Locating the Trace Table

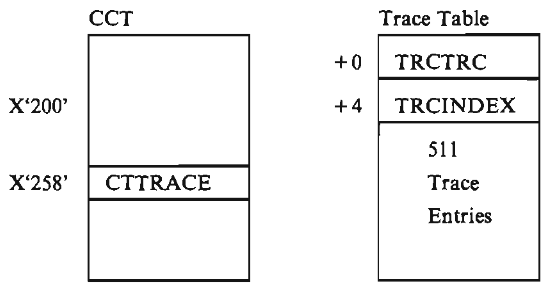


Figure 4-11. Trace Table

The trace table can be located from the CCT by its real address (CTTRACE). The first fullword contains "TRCT" The second fullword (TRCINDEX) contains the index number of the last entry put into the table. This index wraps around from 0 through 510. The remaining portion of the page is an array of doubleword entries.

Entry Formats

- Call Event Record

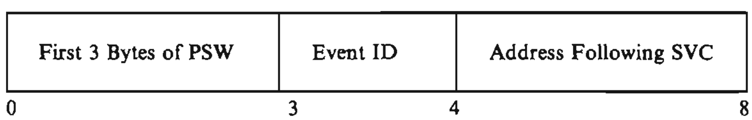


Figure 4-12. Call Event Record

A call event record is the SVC old PSW with the fourth byte replaced by an event ID. The ID is the SVC number of the entry point being called. In the cases where the event is not an SVC, a pseudo-SVC number is assigned, and an appropriate old PSW is used. Note that the DAT mode (first byte sixth bit) must be considered when interpreting the address.

- Return Event Record

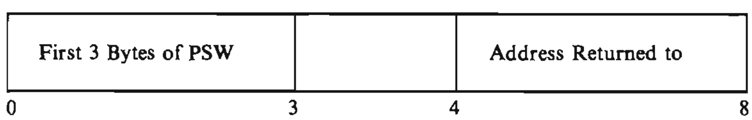


Figure 4-13. Return Event Record

One return event occurs for each call event in the reverse order that the calls occurred. An exception to this is when AMDSAAUD gives control to an error exit. All calls to routines that do not establish their own RCB are eliminated as if the routine, whose RCB is current, had never called them. Note that the DAT mode (first byte sixth bit) must be considered when interpreting the address.

- Error ID Record

00 00 00	0D	Wait Code or Event Address
0	3	4 ... 8

Figure 4-14. Error ID Record.

An error ID record has the same format as an event record except that the first 3 bytes are zeros. The event ID (fourth byte) is X'0D' indicating that AMDSAAUD is the source of the entry. The last 3 bytes identify the error and re the wait-reason code that would be used if SADMP were to terminate.

- Address Fault Record

Pgm Int Code	00	FE	Translation Exception Address
0	2	3	4 ... 8

Figure 4-15. Address Fault Record

An address fault record follows a call event to AMDSAPGI when the program interruption was a translation exception. The ID is X'FE'. The type is indicated by the PI code in the first two bytes, and the last 4 bytes reference the address that caused the exception.

ID	Entry Point Called or Event	Translation Mode	Enablement	Address Space	Reentrancy
00	AMDSASIO	V	D	M	Y
03	AMDSACON	V	D	M	Y
06	AMDSAASM	V	—	M	Y
07	AMDSAERM	V	—	—	N
08	AMDSASIN	V	—	M	N
09	AMDSAUPD	V	D	—	Y
0A	AMDSAGTM	V	D	—	N
0C	AMDSAAID	V	—	M	N
0D	AMDSAAUD or error id record	V	D	—	N
0E	AMDSABUF	V	D	M	Y
0F	AMDSADER	V	D	—	N
10	AMDSAARD	V	—	—	N
11	AMDSAMDM	V	—	L	N
12	AMDSARSM	V	—	—	Y
13	AMDSATER	V	D	—	N
15	AMDSAGTF	V	—	L	N
16	AMDSAUCB	V	—	—	Y
17	AMDSAVCK	V	—	—	Y
18	AMDSABIN	—	—	—	Y
19	AMDSAEBC	—	—	—	Y
1A	AMDSAINR	V	—	—	N
1B	AMDSAPMT	V	E	—	N
1C	AMDSASRR	V	D	M	N
1D	AMDSAEXI	V	D	M	N
1E	AMDSAPGE	V	E	M	N
1F	AMDSAERB	V	—	—	N
20	AMDSAPRS	V	E	—	N
21	AMDSATCP	V	D	—	Y
22	AMDSAT80	V	D	—	N
23	AMDSANTP	V	D	—	N
24	AMDSAXSM	V	—	—	Y
2D	AMDSADOS	R	D	—	N
2E	AMDSADCM	V	D	—	Y
78	AMDSAGET	V	D	—	N
F9	AMDSASVI	V	D	—	N
FA	AMDSAFRM	V	D	—	N
FB	Exit from an interrupt handler	—	—	—	—
FC	Return through SVC linkage	—	—	—	—
FD	AMDSAIOI	V	D	—	Y
FE	AMDSAPGI or Address fault record	R	D	—	N
FF	Machine check	—	—	—	—

Figure 4-16. Trace Event IDs and Entry Point Attributes

The attributes are as follows:

Symbol	Meaning
—	Depends on the caller, or is not applicable
D	Disabled for I/O, external, and machine check interruptions
E	Enabled for I/O, external, and machine check interruptions
L	Local address space being dumped
M	Master address space being dumped
N	No reentrancy
R	Real address translation mode
V	Virtual address translation mode
Y	Reentrancy

How to Read the Trace Data

The events are entered in chronological order. The second word of the table entry is the index of the last entry mode. To get the displacement of the newest entry into the page, multiply the index by 8 (the length of an entry) and add 8 (length of the table header). The fourth byte of each entry is the event ID.

The majority of the event IDs refer to synchronous calls through the SVC linkage interface. These together with ID X'FC', which indicates a return from the most recent call, provide an excellent flow of control trace. Several of the other events represent interruptions that are treated as asynchronous calls. If the wait bit is on (second byte seventh bit) the address will be a wait reason code giving additional information about SADMP.

SVC Linkage Stack

Locating the SVC Stack

The SVC stack can be located from the CCT using its real address (CTSVCSTR) or virtual address (CTSVCSTV). The first fullword contains "SVCS". The second fullword (SSINDEX) contains the index number of the element currently at the top of the stack. The remaining portion of the page is an array that implements the logical stack.. Element 1 is logically the bottom of the stack and element SSINDEX is the top of the stack.

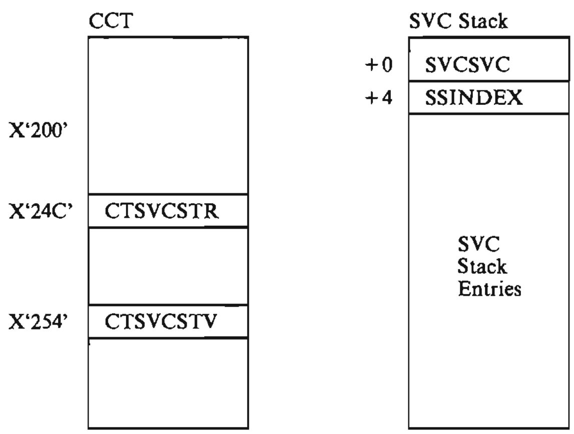


Figure 4-17. SVC Linkage Stack

Use of the SVC Stack

The SVC stack can be located from the CCT either by its real address (CTSVCSTR) or its virtual address (CTSVCSTV). An entry is placed in the stack each time a call is processed by the SVC linkage handler and one is removed each time a return is handled. An entry is added to the stack by increasing SSINDEX. When a recovery event occurs and AMDSAAUD passes control to a recovery routine, SSINDEX is set back down to the value it was when the RCB was enqueued. This ends any chance of activity in any routine called by the routine whose RCB exit will be given control. AMDSAFRM is called to free any short term storage obtained while one of the popped elements was at the top of the stock.

One function of the SVC linkage is to provide a save area for the caller's PSW, STD, and registers. These can be valuable in debugging interface problems.

Storage Assignment (Static and Dynamic) Control Blocks

DSCEs are of the following types:

In use header — The header DSCE serves as the queue header for those DSCEs that represent storage, but it represents no storage itself. It is marked as representing 0 bytes of storage at location 0. Its real address is at SUT+X'10'.

In use for long-term allocated storage — These DSCEs represent storage that is not normally freed (for example, storage for IODBs and the dump table). They are identified as being on the in-use queue, having a nonzero SVC stack index and a subpool number greater than 127. They can usually be found close to the header DSCE along the long-term queue.

In use for short-term allocated storage — These represent storage that is freed when the program that obtained it finishes (for example, automatic data areas for reentrant modules). They are identified by being on the in-use queue, having nonzero SVC stack index, and a subpool number less than 128. They can usually be found close to the header DSCE along the short-term queue.

In use for unallocated storage — These represent storage that is currently unallocated. They are identified by being on the in-use queue and having an SVC stack index of 0. They are typically intermixed with the other in-use DCSEs.

Available DCSEs — These do not represent storage, but may be used in the future to describe new storage blocks. They are identified by being in the set of available DSCEs, that is implemented as a singly threaded queue headed by SUT+X'14' (real address).

Chapter 5. Spzap (AMASPZAP)

Section 1: Introduction

The AMASPZAP service aid program enables a user to:

- Inspect and modify data in any load module that is a member of a partitioned data set.
- Inspect and modify data in a specific data record which is contained in a direct access data set.
- Dump records from data sets residing on direct access storage devices.

AMASPZAP runs under the operating system. It normally resides in SYS1.LINKLIB; however, it may reside in a password protected private library.

For a description of how to use the AMASPZAP service aid, refer to *Service Aids*.

Section 2: Method of Operation

The diagrams in this section show the functions performed by the AMASZAP service aid. SPZAP Figure 5-1 shows the hierarchy of the diagrams and defines the symbols used.

Diagram 1 is an overview of AMASPZAP processing. There are four control statements that determine the path of processing which AMASPZAP will take:

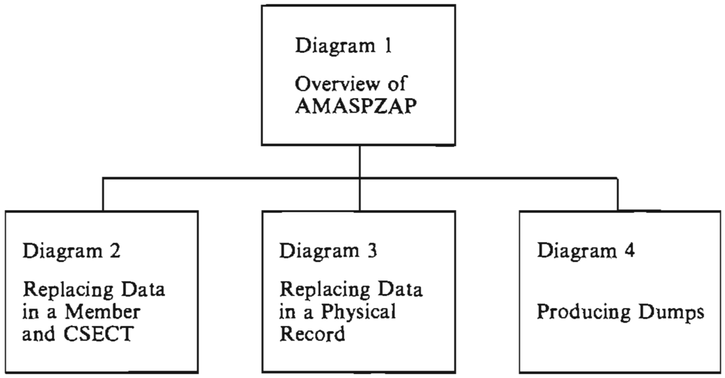
- NAME (Diagram 2) — This statement indicates to AMASPZAP that it will be working with a CSECT contained in a load module which is a member of a partitioned data set. As a part of this processing path, AMASPZAP may replace data in the CSECT, verify the data to be replaced, and/or update SSI and IDR records.

- CCHHR (Diagram 3) — This statement indicates to AMASPZAP that it will be working with a physical record contained in a direct access data set. As a part of this processing path, AMASPZAP may replace data in the record and verify the data to be replaced.

- DUMP(T) or ABSDUMP(T) (Diagram 4) — These statements indicate to AMASPZAP that it is to dump a specific control section or all CSECTs in a load module (DUMP), that it is to dump an entire data set, a member of a partitioned data set, or a groupd of records residing in a direct access data set (ABSDUMP).

Reading Method of Operation Diagrams

Method of operation diagrams are arranged in an input-processing-output layout: the left side of the diagram contains the data that serves as input to the processing steps in the center of the diagram, and the right side contains the data that is output from the processing steps. Each processing step is numbered; the number corresponds to the verbal description of the step in the extended description. While the processing step in the diagram is in general terms, the corresponding text is a specific description that includes a cross-reference to the code for the processing.



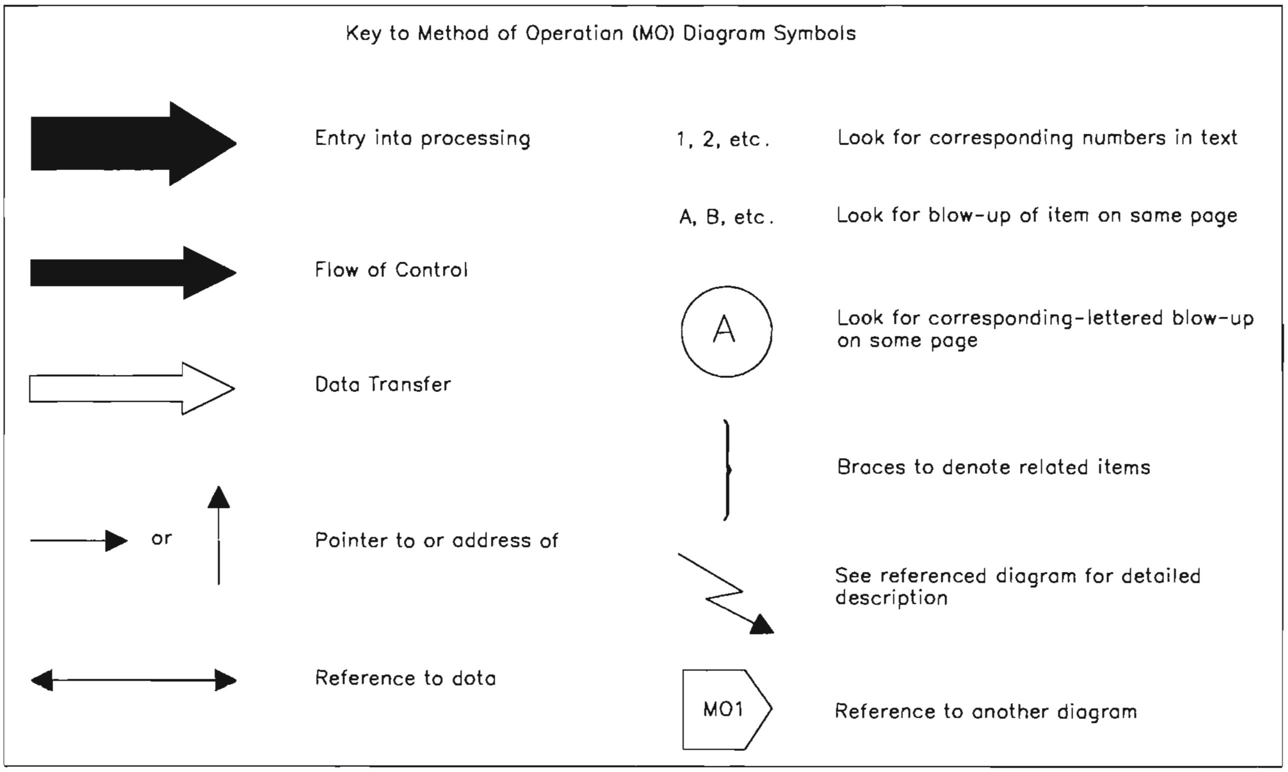
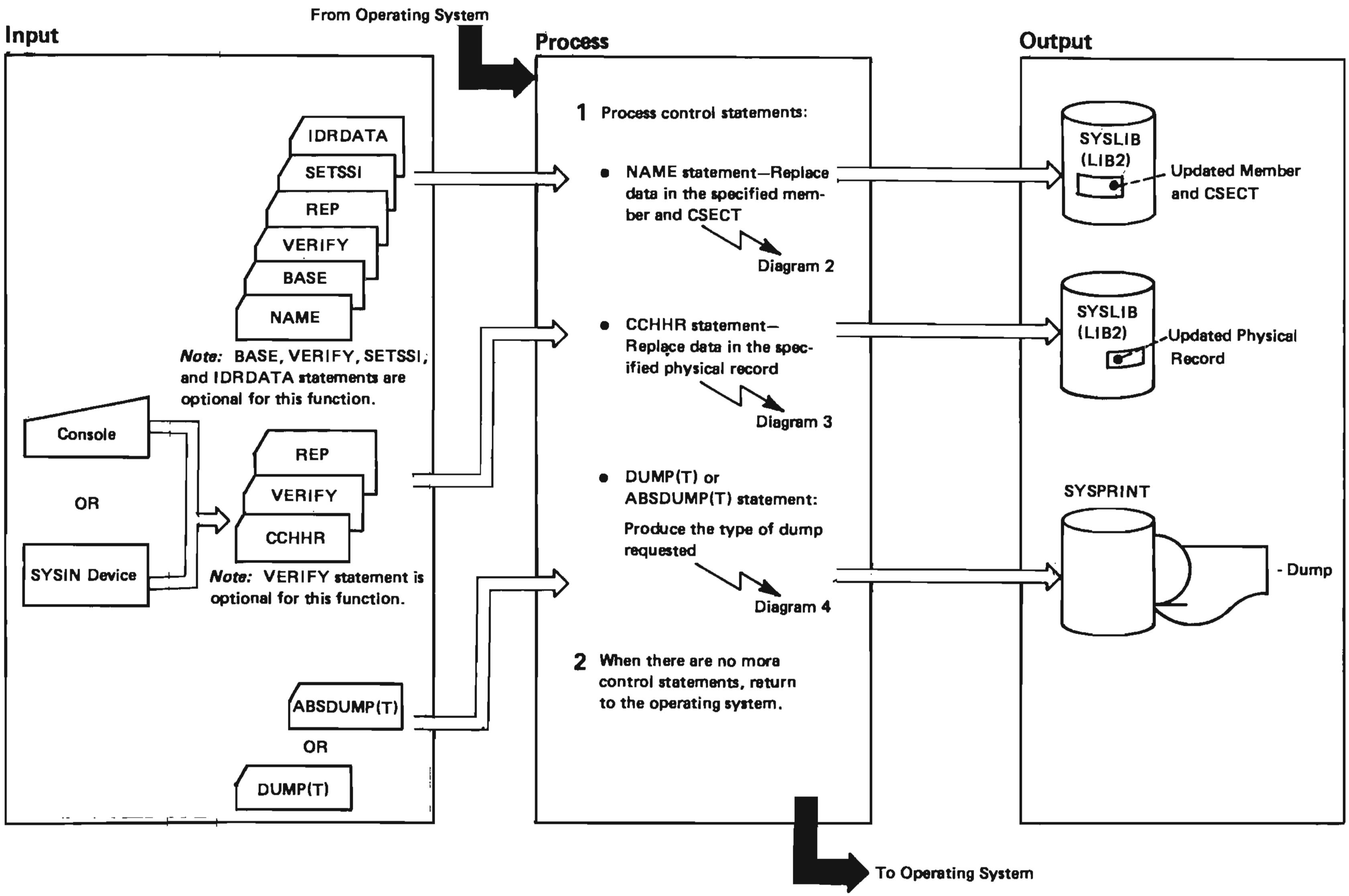


Figure 5-1. Key to Method of Operation Diagrams for AMASPZAP

SPZAP Diagram 1. Overview of AMASPZAP



This page left blank intentionally.

SPZAP Diagram 2. Replacing Data in a Member and CSECT (Part 1 of 4)

Input

NAME Member [CSECT]

SYSLIB (LIB2)
- CESD
- IDR
- CONTROL
- TEXT
- CONTROL
- TEXT

BASE Offset

VERIFY Offset Expected Content

REP Offset Data

Process

1 Find the member and CSECT on SYSLIB. Save the external symbol dictionary ID (ESDID), which was found in the composite external symbol dictionary (CESD) entry for the CSECT.

2 Position to the text record which contains the CSECT, save CSECT address.

3 Save the specified offset.

4 Read the specified record and determine if it contains the expected data:

- Yes – continue processing cards.
- No – ignore subsequent REP card dump erroneous data. MO 4.2

5 Replace (patch) the specified record providing that the verify operation was successful or that there was no VERIFY statement. Indicate that operation has been performed.

MO 1

Output

CSECTID: ESDID

IDSAVE: CSECT Address

BASE2: Offset

SYSLIB Record[1] RDAREA 1

IGNOP: 1

SYSLIB (LIB2): Updated CSECT

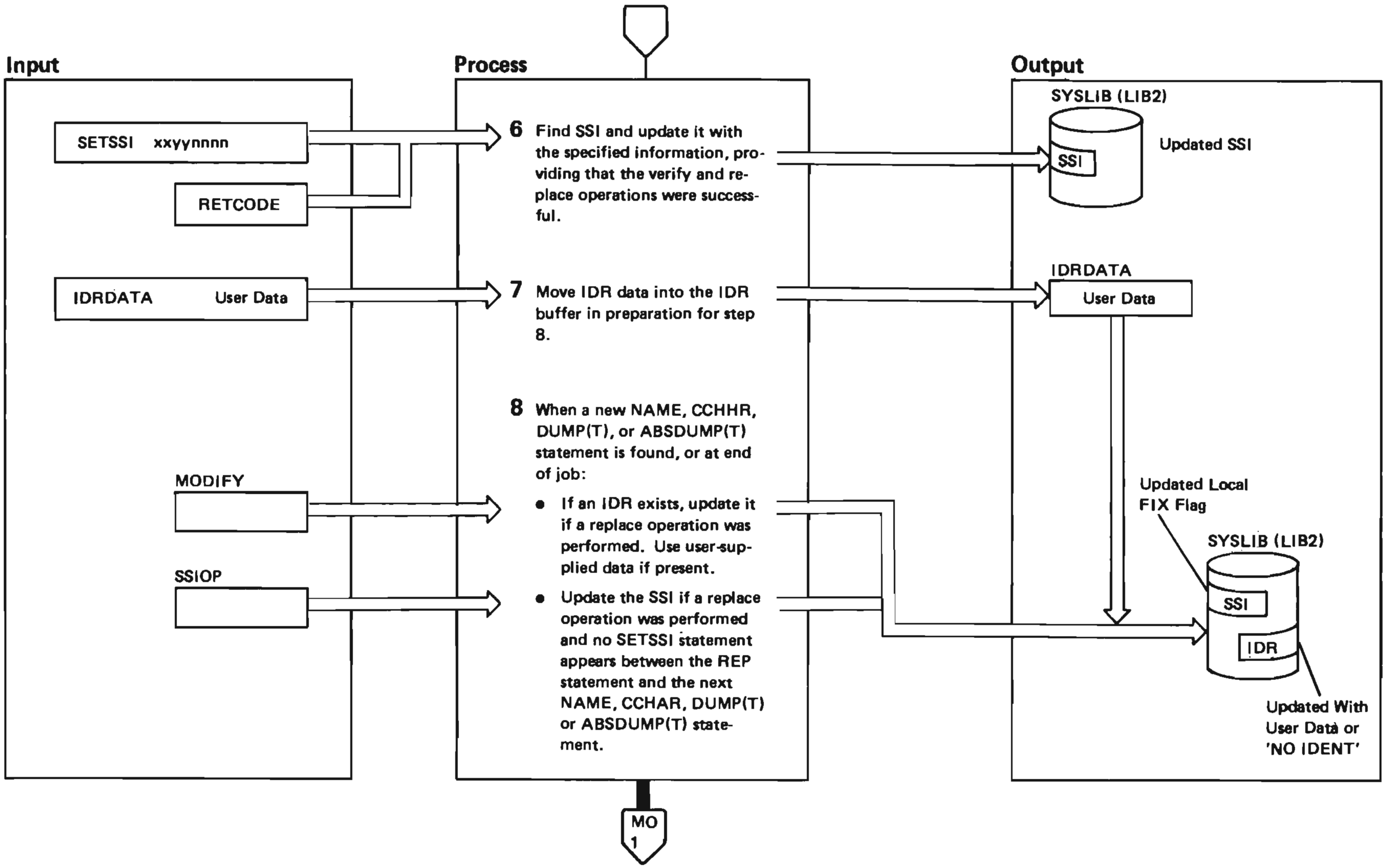
MODIFY

CTLBTE2

SPZAP Diagram 2. Replacing Data in a Member and CSECT (Part 2 of 4)

Explanation	Module	Label
1 A BLDL macro instruction is issued to determine if the member exists. If it does and a CSECT name is specified, the routine searches the CESD entries for the CSECT name. If no CSECT name is specified, find the first (lowest address) CSECT in the member.	AMASPZAP	NAMERTN NAME01A
2 Position to the text record containing the CSECT by matching the ESDID with the ID in the control record which precedes the text record. Add the appropriate offset to obtain the CSECT start address. If the scan finds at least one SPZAP IDR record but no unoccupied IDR entries, the return code will be set to 8 and subsequent REP and IDRDATA statements will be ignored, unless the IGNIDRFULL option has been specified.		NAME03D RDDSK4 SPZAPIDR NAME04
3 If a BASE statement is read, save the specified offset so that it may be subtracted from the offsets given in subsequent REP and VERIFY statements.		BASERTN
4 If a VERIFY statement is read, find the record containing the data to be verified, determine the absolute address of the data, then read and compare the data. A split verify is made when the data to be verified extends over more than one text record. If the comparison is negative, do not perform a replace operation and dump the incorrect data.		VERYRTN SETUP1 NAME07 CCHNOTE NAME13 DUMP01A
5 Find the record containing the data, determine the absolute address of the data, and then make the replacement (PATCH). A split is made when the data to be replaced extends over more than one text record. The MODIFY bit is set to 1 when the replacement is completed successfully.	AMASPZAP	REPRTN SETUP1 NAME07 CCHNOTE

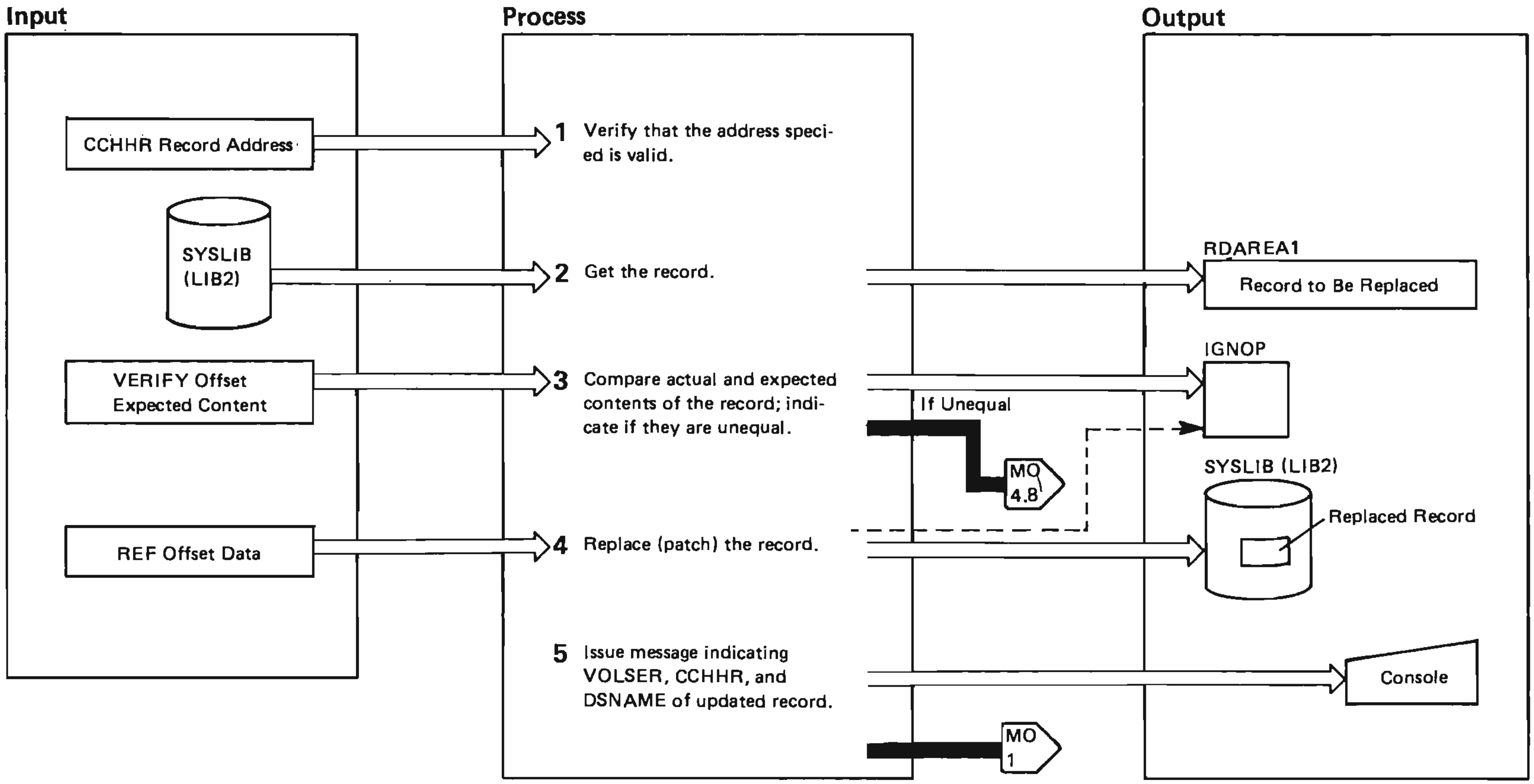
SPZAP Diagram 2. Replacing Data in a Member and CSECT (Part 3 of 4)



SPZAP Diagram 2. Replacing Data in a Member and CSECT (Part 4 of 4)

Explanation	Module	Label
6 If a SETSSI statement is read and there have been no previous errors, the address of the SSI is obtained and placed in register 4. The user-specified SSI is then moved to the SSI.		SSIRTN INSM1A
7 If an IDRDATA statement is read and there is an unoccupied SPZAP IDR entry for the module, the IDRDATA value is stored in the user-data field of the IDR entry (which otherwise will be set to "NO IDENT"). If there are no empty IDR entries, or no IDR records, IDRDATA statements are ignored.		IDRRTN
8 The IDR record is updated with the contents of the IDR DATA field (Either user data or "NO IDENT"). A message is issued giving the current and maximum number of IDR entries. (If current-maximum, an additional message is issued.) If the SSI is to be updated, the local FIX flag in the SSI is set.		INSMDFY
Note: If a CHECKSUM statement is read, the contents of the checksum accumulator are converted to printable hexadecimal, and the accumulator is reset. If the CHECKSUM statement has an operand equal to the converted checksum value, a checksum-correct message is written. Otherwise, the IGNOP bit of CTLBTE is set to inhibit REPs and SETSSIs until the next NAME or CCHHR statement.	AMASPZAP	CHSUMRTN CHSUM

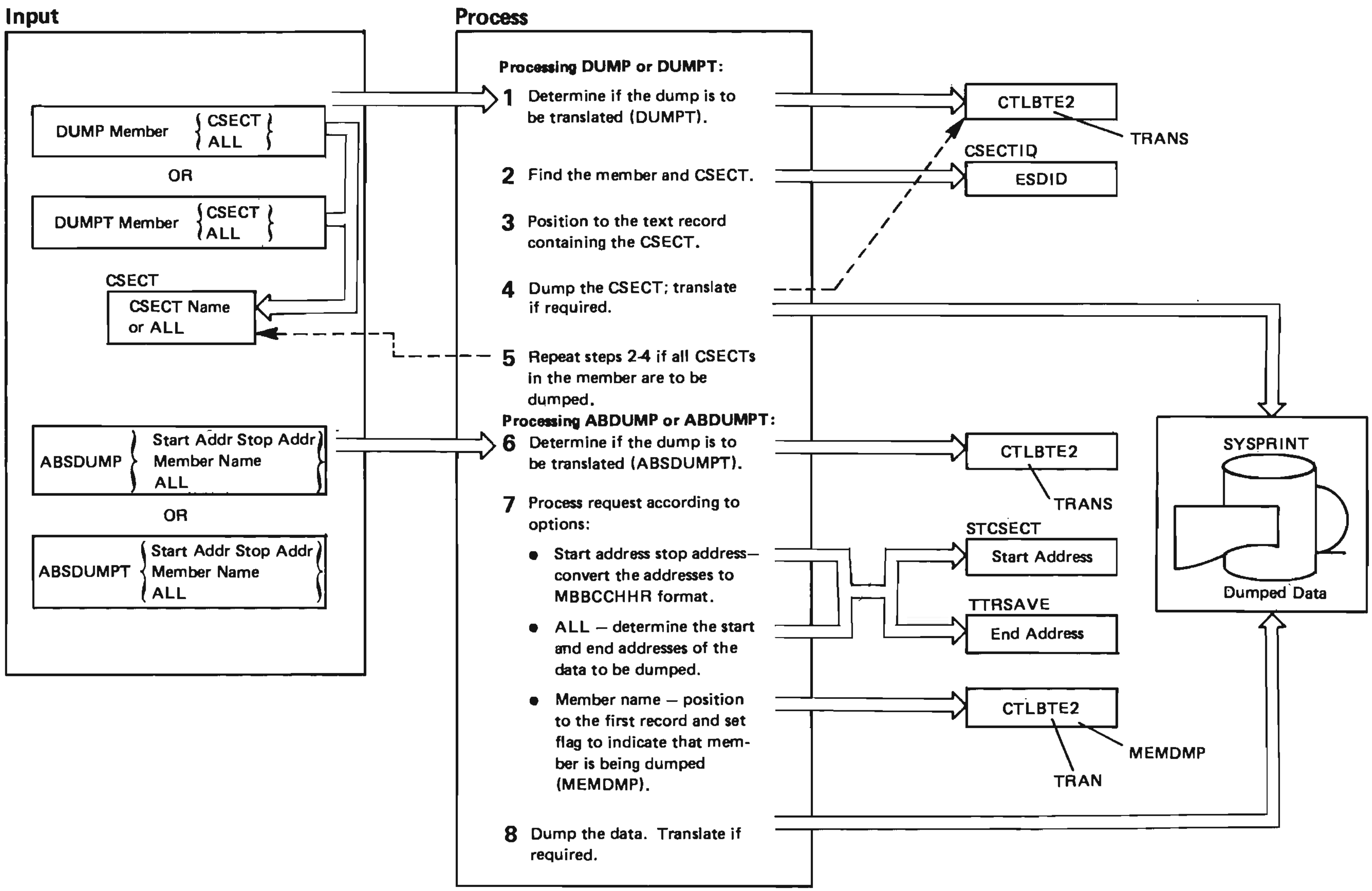
SZAP Diagram 3. Replacing Data in a Physical Record (Part 1 of 2)



SPZAP Diagram 3. Replacing Data in a Physical Record (Part 2 of 2)

Explanation	Module	Label
1 The address specified in the CCHHR statement must be within the limits of the data set specified on the SYSLIB DD statement.	AMASPZAP	CCHHRTN CCHHSUB
2 The record is read from the SYSLIB data set to RDAREA1.		RDDSK
3 If the actual and expected contents do match, the next record is read. Otherwise, the IGNOP switch is set so that a replace operation is not performed, and a dump of the unmatched record is taken.		VERYRTN SETUP1 DUMP01AB
4 Providing that the verify operation was successful or if there was no VERIFY statement, the replacement is made.		REPRTN SETUP1
5 Message AMA121I is issued if a replace operation has been performed when a new NAME, CCHHR, DUMP(T), or ABSDUMP(T) statement is read, or, at end of job.		INSMDFY

SPZAP Diagram 4. Producing Dumps (Part 1 of 2)



SPZAP Diagram 4. Producing Dumps (Part 2 of 2)

Explanation	Module	Label
1 The TRANS switch in CTLBTE is set if DUMPT was specified.	AMASPZAP	DUMPRTN
2 A BLDL macro instruction is issued to determine if the module exists. If it does, the routine searches the CESD (composite external symbol dictionary) entries for the CSECT name.		DUMPR1 DUMP04 NAME01A
3 Match the ESDID with the ID in the control record which precedes the text record. Add the appropriate offset.		DUMP01B DUMP06B RDDSK4
4 The CSECT records are read, formatted, and written to the SYSPRINT data set. If DUMPT was specified, the records are translated before being written. The formats of translated and untranslated dumps are shown in *Service Aids*.		DUMP1 DUMP01 GRAP1
5 If ALL was specified, processing continues until all CSECTs in the member have been dumped.		DUMP04 DUMP06B
6 The TRANS switch is set if ABSDUMPT was specified.	AMASPZAP	ABDMPRTN
7 When the start address and stop address are specified on the control card, the addresses are converted and moved into the STCSECT and TTRSAVE fields.		ABDMP1B ABDMP2 CHCONV
When ALL is specified on the control card, the start address of the SYSLIB data set is determined by examining the DEB. The end address of the data set is determined by examining the VTOC via the OBTAIN macro.		ABDMP3 CHCONV
When a member name is specified on the control card, the routine gets the member name and issues the FIND macro instruction to position to the first record. The MEMDMP switch is set to indicate that the dump operation should be stopped the first time end-of-file is reached.		ABDMP6 SCANKEYS

Explanation	Module	Label
8 A dump of the specified data is produced. If ABSDUMPT was specified, the records are translated before being written. The formats of translated and untranslated dumps are shown in *Service Aids*.		
Start & Stop adr & ALL		ABDMP5 DUMP1 DUMP01 CONTB
Member		DUMP1 DUMP01
If translate		GRAP1

Section 3: Program Organization

The superzap service aid program consists of one module which contains four CSECTs:

- AMASPZAP — controls the service aid operations and routes control to the AMASZDMP and AMASZIOR CSECTs.
- AMASPZAP — dumps data to the SYSPRINT data set.
- AMASPZAP — defines the constants used by the AMASPZAP service aid program.
- AMASPZAP — performs input/output and exit processing.

CSECT Descriptions

AMASPZAP — Control CSECT

Entry from: The operating system.

Entry point names: AMASPZAP.

Data areas defined or updated: None.

Size: 4K

Routines called: AMASZIOR, AMASZDMP.

Exits: To AMASZIOR.

Operation: Routes control between subroutines in AMASZDMP and AMASZIOR; controls the processing.

AMASZCON —Superzap Constants CSECT

Entry from: N/A

Entry point names: N/A

Data areas defined or updated: N/A

Size: 3K.

Routines called: N/A

Exits: N/A

Operation: Contains constants and data areas used by the other three superzap CSECTs.

AMASZDMP — Dump CSECT

Entry from: CSECT AMASPZAP.

Entry point names: AMASZDMP.

Data areas defined or updated: None.

Size: 1K.

Routines called: AMASZIOR.

Exits: Return to AMASPZAP.

Operation: Performs DUMP and ABSDUMP processing.

AMASZIOR — Input/Output and Exit CSECT

Entry from: AMASPZAP, AMASZDMP.

Entry point names: AMASZIOR.

Data areas defined or updated: None.

Size: 2K.

Routines called: None.

Exits: Return to caller or exit to supervisor.

Operation: Performs I/O operations, issues WTO macro instructions, contains SYNAD routines, and performs clean-up for exit processing.

Section 4: Diagnostic Aids

This section contains the following information to aid the reader in diagnosing SPZAP errors:

- A list of SPZAP switches that control processing.

Notes:

1. *SPZAP consists of one module (AMASPZAP) with four CSECTs; AMASPZAP, AMASZCON, AMASZDMP, and AMASZIOR.*

2. *Refer to System Messages for message numbers, message text, and detecting, issuing, and containing modules.*

AMASPZAP Switches

The following defines the switches that AMASPZAP uses to control processing.

Switch Byte	Switch	Alignment	Description
CTLBTE		x...	Not used.
	NAMEERR	.1..	An error in a NAME or CCHHR statement is detected to prohibit subsequent VER, REP, SETSSI, and IDRDATA operations. The switch is reset when a valid NAME or CCHHR statement is found.
	ENDMOD	..1.	The last text record of a load module has been read.
	OVFLOW	...1	Data to be verified or replaced extends over more than one block of text.
	VERYOP	.... 1...	A verify operation is being performed.
	IGNOP	.... .1..	A verify operation is unsuccessful. Reset when a NAME, CCHHR, DUMP, or ABSDUMP statement is read.
	NAMEOP	.... ..1.	A NAME statement is read to indicate that subsequent REP and VERIFY statements are for a CSECT within a load module.
	ONEREC	1	A load module contains no CESD records.
CTLBTE2	PSIND	1...	ABSDUMP(T) ALL has been specified for a sequential data set. It indicates that the dump should be terminated when end of the file (EOF) is reached.
		.xx.	Not used.
	FIRSTPCH	...1	More data may need to be moved into the text of message AMA122I if the data to be replaced extends over more than one text record.
	MEMDMP	.... 1...	A member name is specified in an ABSDUMP statement.
	MULREC	.... .1..	A load module is contained in more than one text record.
	TRAN	.... ..1.	The translate (T) option was specified on a DUMP or ABSDUMP statement.
	MODIFY	1	A replace operation has been performed for a CSECT specified in a NAME statement (contrast with MODFYREC).
	CTLBTE3	SSIOP	1... .... An SSI update requested by a SETSSI card is being performed. If this switch is off and a replace operation has been performed, AMASPZAP sets the local fix flag in the SSI.
	NAMEIDR	.1..	An IDRDATA statement is read.
	IDRRD	..1.	IDR space exists for new CSECT update (NAME and REP) entry.
	MODFYREC	...1	A replace operation has been performed at the address specified in a CCHHR statement (contrast with MODIFY).
	NOSSI	.... 1...	There is no SSI for the module to be updated.
	NOMEM	.... .1..	The name of the member whose SSI is to be updated is not found in the PDS directory.
	LMREAD	.... ..1.	All CESD records have been read to indicate that the input buffer (RDAREA1) contains an IDR or control record.
	NENOIDR	1	A load module is not editable and contains no CESD records.

Index

MVS/Extended Architecture Service Aids Logic

LY28-1189-2

S370-37

MVS/Extended
Architecture
Service Aids Logic

LY28-1189-2

READER'S
COMMENT
FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

What is your occupation? ________________________________

How do you use this publication? ________________________________

Number of latest Newsletter associated with this publication: ________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

MVS/Extended Architecture Service Aids Logic

Contains Restricted Materials of IBM
Licensed Materials - Property of IBM

LY28-1189-2 S370-37

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

Printed in U.S.A.

IBM®

MVS/Extended
Architecture
Service Aids Logic

LY28-1189-2

READER'S
COMMENT
FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

What is your occupation? ______

How do you use this publication? ______

Number of latest Newsletter associated with this publication: ______

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

MVS/Extended Architecture Service Aids Logic

LY28-1189-2

S370-37

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

Printed in U.S.A.

IBM®

MVS/Extended
Architecture
Service Aids Logic

LY28-1189-2

READER'S
COMMENT
FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

What is your occupation? ______________________________

How do you use this publication? ______________________________

Number of latest Newsletter associated with this publication: ________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

MVS/Extended Architecture Service Aids Logic

Contains Restricted Materials of IBM
Licensed Materials - Property of IBM
(Except for Customer-Originated Materials)

LY28-1189-2

S370-37

Reader's Comment Form

Cut or Fold Along Line

Fold and tape   Please Do Not Staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape   Please Do Not Staple   Fold and tape

Printed in U.S.A.

IBM®

LY28-1189-02