amdahl®

MVS/SP Assist
Release 1.0
Software Logic Manual

amdahl®

MVS/SP Assist Release 1.0 Software Logic Manual

Publication Number: L1020.0-02A

October 1982

REVISION NOTICE
This is the second edition. It replaces L1020.0-01A, September 1982. It incorporates new technical information shown with revision bars.

ABSTRACT
This manual provides technical information for the Amdahl MVS/SP Assist (MVS/SPA) Release 1.0 program product (4PZ0-C3-U). It is intended for systems programming and support personnel who are responsible for problem determination and diagnosis of MVS/SP Assist problems. The *MVS/SP Assist General Information Manual* (Amdahl M1130.0) is recommended as a prerequisite.

READER COMMENT FORM
A reader comment form is provided at the end of this manual. If this form is not available, comments and suggestions may be sent to Amdahl Corporation, Technical Publications Department, Mail Stop 323, P.O. Box 470, Sunnyvale, CA 94086. All comments and suggestions become the property of Amdahl Corporation.

CONTENTS

FIGURES

TABLES

CHAPTER 1 - INTRODUCTION

MVS/SP Assist (MVS/SPA) Release 1.0 is an Amdahl licensed program product that allows the MVS/System Products (5740-XYN and 5740-XYS) at Version 1 Release 3 to run on Amdahl 470 processors and System/370 uniprocessors that do not have the 370 Extended Facility/Feature (EF) installed. In addition, it provides support for the 3033 Extension (XF) for those 470, 370, 303X and 4300 systems that do not have the 3033 Extension available or installed. It also provides support for Extended Addressing (EA) on the Amdahl 470 systems. See table 1-1 for supported combinations.

| MVS/SPA supersedes MVS/SEA Release 3.0 and, because of a different design, does not require most of the operating system changes required by MVS/SEA; therefore the installation procedure is much simpler. It gives additional performance gains for an MVS/SP 1.3 system that currently uses MVS/SEA and/or the IBM-provided XF simulation routines.

MVS/SPA will produce results functionally equivalent to any of the following environments:

MVS/SP 1.3 with IBM XF simulations and MVS/SEA

MVS/SP 1.3 with IBM XF simulations and 370/EF

MVS/SP 1.3 with the 3033 Extension

Unsupported features and exceptions to the operation of the supported instructions, as defined in the _System/370 Principles of Operation_ or _System/370 Assists for MVS_, are listed in Appendix E.

NOTE

As is the case with MVS/SEA, MVS/SPA assumes that it will be the only user of the unused sections of the PSA. Should any other user installed products also require use of some or all of this area, please see paragraph 3.7 for information on how to resolve these conflicts.

1.1 RELATED DOCUMENTATION

The documents listed below provide the user with additional information about the Amdahl MVS/SPA and MVS/SEA products, and the IBM 370/Extended Facility, the 3033 Extension and Extended Addressing.

IBM System/370 Principles of Operation (IBM GA22-7000)

IBM System/370 Assists for MVS (IBM GA22-7079)

Cross Memory Services User's Guide (IBM GG22-9231)

MVS/SP Assist Release 1.0 General Information Manual (Amdahl M1130.0)

MVS/SE Assist Release 3.0 General Information Manual (Amdahl M1109.0)

Amdahl MVS/SPA Microfiche (Amdahl Licensed Material)

Table 1-1. Supported Systems and Features

SYSTEM	370/EF	3033E	EA
Amdahl 470	X	X	X
IBM 370 w/o EF	X	X	
IBM 370 w/EF		X	
IBM 303x w/o 3033E		X	
IBM 303x w/3033E	Not Used - Completely Transparent		
IBM 4300 w/o ECPS:MVS	X	X	
IBM 4300 w/ECPS:MVS		X	
IBM 4300 w/3033E EC	Not Used - Completely Transparent		
Other CPUs		As Required	

CHAPTER 2 - OVERVIEW

A CPU running MVS/SP 1.3 without the 3033 Extension will program check when it encounters one of the XF instructions. This program check is passed to nucleus-resident module IEAVEXMS to see if the instruction is indeed one of the XF instructions and not a coding error. If it is an XF instruction, it is simulated, and return is made either to the next sequential instruction or otherwise as the instruction dictates.

The MVS/SPA product takes advantage of this design to also intercept the program checks from the EF and EA instructions; module IEAVEXMS is completely replaced, and SPA simulates or traps all three types of instructions. The simulations are faster in most cases than those in the standard IEAVEXMS, and the construction of the MVS/SPA IEAVEXMS eliminates the possibility of a program check recursion caused by the issuance of EF instructions in the standard IEAVEXMS. In some cases the existence of an MVS coding convention is checked for, allowing a fast path in the decoding and fetching of operands - for example, in the Program Call and Program Transfer simulations.

Normally, all EF (and some XF) instructions are replaced by valid instructions that either directly enter the simulation routines or simulate the original instruction.

For program interruptions that are not related to EF, XF or EA instructions, MVS/SPA passses control back to the program check FLIH to continue with normal RTM processing. Also, should a simulation routine encounter a cause for exception while simulating an instruction, the exception is passed to the program FLIH as it would have been had the hardware feature been installed.

The MVS/SPA product consists of one module, a replacement for IEAVEXMS. In addition, a small superzap change to module IEAVNIPO is required for CPUs without the 370/EF implemented in hardware to allow MVS/SP to IPL without it. On 370/168 CPUs without EF, or under non-SP VM systems, a change is also required to IEAVNPX1 to ensure that the common segment bits are not turned on, as the MVS/SPA product cannot intercept the resulting program checks that would result. See appendix D for further information.

The 470 Extended Addressing feature requires some additional changes to the operating system. The feature is checked for proper enablement by a zap to IEAVNIPO, and three standalone dump modules are updated to support the Amdahl implementation of the extended key instructions. Paragraph 3.3 describes the necessary feature settings to enable the 470 Extended Memory feature after it has been installed.

Appendix F summarizes the operating system modules changed and the nature of the changes.

CHAPTER 3 - METHOD OF OPERATION

Flowcharts for most of the MVS/SP Assist routines are provided in chapter 4.

Module IEAVEXMS is passed control from the Program Check First Level Interrupt Handler (FLIH) whenever a program interrupt occurs for an operation exception (interrupt code X'01') or a privileged operation exception (interrupt code X'02'). The type of interrupt is determined from the program interrupt code (location X'8F' in the PSA), and the instruction address is determined by subtracting the instruction length code (PSA location X'8D') from the address in the program old PSW (PSA location X'28'). The cause of the interrupt will be an operation exception for a supported instruction, a privileged operation exception for an IPK or SPKA, or some other non-supported reason.

Supported instructions are handled in one of three ways: simulating, trapping or ignoring them. In general, EF instructions are trapped, and XF and EA instructions are simulated. The difference between trapping and simulating is that while both result in the instruction being simulated, the trapped instruction uses an exit in the PSA, and operand decodes are already done. The non-trap simulation routines require that the operands be decoded on all occurrences.

3.1 EXTENDED FACILITY/FEATURE (EF) INSTRUCTIONS

When an EF instruction is recognized, the program check old PSW is checked to ensure the issuer was in supervisor state. If not, the interruption code is changed to X'0002' to indicate a privileged operation exception, and control is passed via the error exit to RTM.

The opcode is next checked to ensure that it is within the range X'E501' to X'E50D'. If not, the interrupt is passed back to the program FLIH via the error exit. If the opcode is supported, an address located via table look-up is used to locate the specific simulation routine and to control further processing.

The trace, IPTE and TPROT simulations require trap areas in the PSA. These areas are allocated and the traps are built by the PSAALLOC routine. Appendix A describes this routine in more detail.

3.1.1 Test Protection Simulator - SPAE501

Routine SPAE501 receives control from the PSA exit routine built on the first occurrence of the instruction. On entry, two registers have been loaded with the instruction operands and a third with the routine base register. (See appendix A for details of the PSA exit.)

After setting a recursion indicator into PSAPCFB4, SPAE501 changes the PSW key to that specified in the second operand. The storage address designated by the first operand is then fetched (via a TM instruction) and tested for store permission (via a non-destructive OI). The PSW key is reset to zero to allow resetting of the recursion indicator, and control is returned to the original code via LPSW. The default condition code in the resume PSW is zero (set during trap construction) so the condition code is valid.

If either fetch or store is disallowed, the references by the TM or OI cause a program check. Due to the setting of PSAPCFB4, control is passed to the SPA recursion entry, IEAVEXM2, by the program FLIH. IEAVEXM2 then examines the register contents to decide whether the fetch or the store was disallowed (a register contains 'F' during the fetch trial and 'S' on the store trial). It sets the appropriate condition code in the resume PSW and returns to the original code.

3.1.2 Fix Page Simulator - SPAE502

Routine SPAE502 receives control from the PSA exit routine built on the first occurrence of the instruction. On entry, registers are as follows:

Register 0	- Real address in the first page frame to be fixed.
Register 1	- Virtual address in the first page to be fixed.
Register 2	- Virtual address in the last page to be fixed.
Registers 8-10	- Saved
Register 11	- Routine base register
Register 14	- Pointer to S-type address constant of successful-fix exit point. (Instruction first operand.)

After locating the MAPL, the routine loops, attempting to fix all the pages in the range specified by registers 1 and 2. (The microcode implementation does only one page per issuance of the instruction, but as a performance enhancement, the simulation fixes all possible pages on each issuance.) The following description applies to each pass through the loop.

If the page to be fixed is already fixed by default (nucleus, L/SQA or V=R page), the fix count is not examined, and control passes to the end of the loop.

If the page frame is being fixed for the first time and is not in the preferred area, or if it does not belong to either the common area or a second-level preferred user, the simulation terminates by calling the MVS exception routine, located from MPLPFAL. In all other cases the fix count is incremented for the page. If this is the first fix for the page (fix count is now one), the total system fixed frame count (MPLCNTRS in the PVT) is incremented. If the page is a common area page, the common area fix count is also incremented. If it is a private area page, the address space fix count in the RSM header is incremented. The proper RSMHD is located from the ASCB either in PSA-AOLD (during early NIP), or the ASCB located (via the ASVT) from the ASN in control register 3 or 4, depending on the setting of the secondary mode bit.

If the last page in the range has not yet been fixed, the address of the current page to fix is incremented by X'1000'. If the page exists in storage or the Extended Main Storage feature is not enabled, processing begins again at the start of the loop. If not, the S-constant pointed to by register 14 is decoded and branched to.

If the last page in the range has been fixed, the total system fix count is compared with the maximum number of pages allowed to be fixed contained in the MAPL (MPLMAXFX). If the maximum count has not been exceeded, return is made to the instruction after the FIX PAGE instruction trap. If the count is exceeded, exit is made to the SRM frame fix excession routine, located in the MAPL (MPLPFCM).

3.1.3 SVC Assist Simulator

The function performed by the SVC Assist hardware implementation instruction is identical to that of normal SVC FLIH processing; therefore, there is no SVC Assist simulation routine. Whenever the SVC Assist instruction is encountered, it is overlaid with NOPs so that control passes to standard FLIH processing.

3.1.4 Obtain Local Lock Simulator - SPAE504

Routine SPAE504 receives control directly from the instream code via a LOAD/BALR sequence which replaced the original instruction on first occurrence. Register 12 contains the return address, 13 the base address, and 11 is free. (See appendix A for further details of the linkage.)

First, pseudo-SRB mode (PSAPSRBM) is set to avoid preemption during interrupts, then the PSA super bit for the lock manager (PSALOCK) is set to disable any PER processing that may occur. Because the lock simulators run enabled, these bit-settings prevent any possibility of loss of control during update of the lock word and indicators.

The current ASCB is located via PSAAOLD and the lock status checked. If it is held, exit is made to the 'Local Lock Obtain Failed' entry point in the Lock Interface Table prefix (LITOLOC). If it is not held, the CPU physical address is set into the lock word, the PSAHLHI bit for the local lock is set, and exit is made to the caller. In both cases, the condition code is restored to what it was on entry, and both the SUPER and MODE bits set on entry are reset.

3.1.5 Release Local Lock Simulator - SPAE505

Routine SPAE505 receives control directly from the instream code via a LOAD/BALR sequence which replaced the original instruction on first occurrence. Register 12 contains the return address, 13 the base address, and 11 is free. (See appendix A for further details of the linkage.)

First, pseudo-SRB mode (PSAPSRBM) is set to avoid preemption during interrupts, then the PSA super bit for the lock manager (PSALOCK) is set to disable any PER processing that may occur. Because the lock simulators run enabled, these bit-settings prevent any possibility of loss of control during update of the lock word and indicators.

The 'locks held' bit string is checked to ensure that a CMS lock is not held and the local lock is held. The current ASCB is located via PSAAOLD and the local lock suspend queue is checked. If a CMS lock is held, the local lock is not, or there is an entry in the suspend queue, exit is made to the 'Local Lock Release Failed' entry point in the LIT prefix (LITRLOC). If all the checking is passed, the lock-word is zeroed, the lock held indicator (PSAHLHI) is reset, and exit is made to the caller. In both cases, the condition code is restored to what is was on entry, and both the SUPER and MODE bits set on entry are reset.

3.1.6 Obtain CMS Lock Simulator - SPAE506

Routine SPAE506 receives control directly from the instream code via a LOAD/BALR sequence which replaced the original instruction on first occurrence. Register 12 contains the return address, 13 the base address, and 11 points to the CMS lockword to be used. (See appendix A for further details of the linkage.)

First, pseudo-SRB mode (PSAPSRBM) is set to avoid preemption during interrupts, then the PSA super bit for the lock manager (PSALOCK) is set to disable any PER processing that may occur. Because the lock simulators run enabled, these bit-settings prevent any possibility of loss of control during update of the lock word and indicators.

The specified lockword is first inspected to ensure that it is free. If it is not, or a CMS lock is already held, or the local lock is not, exit is made to the 'CMS Lock Obtain Failed' entry point in the Lock Interface Table prefix (LITOCMS). Field PSALOCAL is then used to locate the proper ASCB (if it is zero, PSAAOLD is used). The CMS lock held indicator is then turned on in the PSAHLHI field, and the ASCB address is stored into the lock to mark it held. Exit is then made to the caller. In all cases, the condition code is restored to what it was on entry, and both the SUPER and MODE bits set on entry are reset.

3.1.7 Release CMS Lock Simulator - SPAE507

Routine SPAE507 receives control directly from the instream code via a LOAD/BALR sequence which replaced the original instruction on first occurrence. Register 12 contains the return address, 13 the base address, and 11 points to the lockword to be used. (See appendix A for further details of the linkage.)

First, pseudo-SRB mode (PSAPSRBM) is set to avoid preemption during interrupts, then the PSA super bit for the lock manager (PSALOCK) is set to disable any PER processing that may occur. Because the lock simulators run enabled, these bit-settings prevent any possibility of loss of control during update of the lock word and indicators.

The 'locks held' bit string is checked to ensure that both a CMS lock and the local lock are held. If either lock is not held, or there is an entry in the suspend queue, exit is made to the 'CMS Lock Release Failed' entry point in the LIT prefix (LITRCMS). If all the checking is passed, the lockword is zeroed, the lock held indicator (PSAHLHI) is reset, the condition code is restored to what it was on entry, both the SUPER and MODE bits are reset, and exit is made to the caller.

3.1.8 Trace Instruction Simulations - SPAE508,9,A,B,C,D

The method of operation is the same for all of the trace instruction simulations: the simulation routine is entered from a unique PSA trap that disables the machine, saves some registers, moves the model resume PSW to the resume area (PSAXMPSW) and calls the simulation routine. (See appendix A for further details of the linkage.)

The trace table header is first checked to see if the table has wrapped. If so, the table start address is used for the new entry address and condition code 1 is set in the resume PSW. Otherwise the current entry pointer is incremented by 32, and the new entry address is stored back into the current entry pointer.

The format of each entry is defined in IBM System/370 Assists for MVS. The only difference is that in some cases an operand is assumed and fetched from the PSA, instead of decoding the instruction operand and using that address. Results should be identical in both cases. Figure 3-1 contains a layout of the trace entries.

The specific trace routines are:

- SPAE508 - Trace SVC Interrupt
- SPAE509 - Trace Program Interrupt

INSTRUCTION		TRACE SVC INTERRUPTION	TRACE PGM INTERRUPTION	TRACE INITIAL SRB DISPATCH	TRACE I/O INTERRUPTION	TRACE TASK DISPATCH	TRACE SVC RETURN	TRACE PROGRAM CALL	TRACE PROGRAM TRANSFER	TRACE SET SECONDARY ASN
ROUTINE NAME		SPAE50B	SPAE50B	SPAE50A	SPAE50B	SPAE50C	SPAE50D	SPAB213A	SPAB223A	SPAB225A
BYTES 0, 1		FLCSOPSW BYTES 0, 1	FLCPOPSW BYTES 0, 1	PSASMPSW BYTES 0, 1	FLCIOPSW BYTES 0, 1	PSAPSWSV BYTES 0, 1	PSAPSWSV BYTES 0, 1	NEW PSW BYTES 0, 1	NEW PSW BYTES 0, 1	NEW PSW BYTES 0, 1
BYTE 2	BITS 0 – 3	"2"	"3"	"4"	"5"	"7"	"8"	"9"	"A"	"B"
	BITS 4 – 7	"0"	"0"	"0"	FLCIOAA BITS 12 – 23	"0"	"0"	"0"	"0"	"0"
BYTE 3		FLCSVCN BYTE 1	FLCPICOD BYTE 1	PSASMPSW BYTE 3		RBINTCOD BYTE 1	RBINTCOD BYTE 1	NEW PSW BYTE 3	NEW PSW BYTE 3 – 7	NEW PSW BYTE 3–7
BYTES 4 – 7		FLCSPPSW BYTES 4 – 7	FLCPOPSW BYTES 4 – 7	PSASMPSW BYTES 4 – 7	FLCIOPSW BYTES 4 – 7	PSAPSWSV BYTES 4 – 7	PSAPSWSV BYTES 4 – 7	NEW PSW BYTES 3–7		
BYTES 8, 9		GENERAL REGISTER 15	GENERAL REGISTER 15	"0"	FLCCSW	GENERAL REGISTER 15	GENERAL REGISTER 15	NEW PASN	NEW PASN	PASN
BYTES 10, 11				SRBPASID				NEW SASN	"0"	NEW SASN
BYTES 12 – 15		GENERAL REGISTER 0	FLCTEA	GENERAL REGISTER 0		GENERAL REGISTER 0	GENERAL REGISTER 0	GENERAL REGISTER AFTER 14	OLD PASN / "0"	"0" / OLD SASN
BYTES 16 – 19		GENERAL REGISTER 1	GENERAL REGISTER 1	GENERAL REGISTER 1	"0"	GENERAL REGISTER 1	GENERAL REGISTER 1	"0"	"0"	"0"
BYTE 20	BITS 0 – 1	FLCSILC BITS 5, 6	FLCPILC BITS 5, 6	"0"	FLCIOPSW BYTE 2	RBINLNTH BITS 5, 6	RBINLNTH BITS 5, 6	"10"	"10"	"10"
	BITS 2 – 7	FLCSOPSW BITS 18 – 23	FLCPOPSW BITS 18 – 23			PSAPSWSV BITS 18 – 23	PSAPSWSV BITS 18 – 23	FLCSOPSW BITS 18–23		
BYTE 21		PSACPUSA+1								
BYTES 22, 23		ASCBASID	ASCBASID	ASCBASID	ASCBASID	ASCBASID	ASCBASID	"0"	"0"	"0"
BYTES 24 – 27		PSATOLD	PSATOLD	SRBPTCB	PSATOLD	PSATOLD	PSATOLD	PC #	"0"	"0"
BYTES 28 – 31		BYTES 2–6 OF DOUBLEWORD THAT IS STORED BY INSTRUCTION "STORE CLOCK"								

A03103

Figure 3–1. Trace Table Entry Formats

- SPAE50A - Trace Initial SRB Dispatch
- SPAE50B - Trace I/O Interrupt
- SPAE50C - Trace Task Dispatch
- SPAE50D - Trace SVC Return

3.1.9 Invalidate Page Table Entry Simulation - SPAB221

Routine SPAB221 receives control from the PSA exit routine built on the first occurrence of the instruction. On entry, two registers have been loaded with the instruction operands and a third with the routine base register. The fourth register has been loaded by a BAL that calls the simulation routine, to provide the condition code on entry. (See appendix A for details of the PSA exit.)

A PURGE TLB (PTLB) is done immediately upon entry to allow the purge to execute asynchronously. After setting the recursion indicator (PSAPCFB4), input operand values are used to locate the proper page table entry to be marked invalid. If the issuer of the instruction was not in key zero, a SPKA is done to enter user key, and the page table entry is marked invalid. After housekeeping, return is made directly to the caller via the resume PSW.

3.2 3033 EXTENSION (3033E,XF) SUPPORTED INSTRUCTIONS

All of the 3033 Extension supported instructions have X'B2' opcodes. After housekeeping at entry point IEAVEXM1, the second byte of the opcode is retrieved, validity checked (it must be between X'21' and X'2B'), multiplied by 4 and used as an index into a table. The table contains the addresses of the various simulation routines for the B2 opcodes.

3.2.1 Program Call Simulator - SPAB218A

After setting the recursion indicator, SPA checks to ensure the caller has translate on and is not in secondary mode. The instruction operand is then decoded, with a fast path if the operand is '0(R2)', the MVS convention. If the fast path is not used, the base register (if any) is retrieved and the displacement (if any) added in.

The program call number (PCN) is now translated, in a process very similar to virtual address translation. After ensuring that PCN translation is allowed (bit 8 of CR5), the Linkage Index (LX) is isolated from bits 17-23 of the PCN, and used as an index to the proper Linkage Table Entry to retrieve the Entry Table Origin. The low-order byte of the PCN is used as an Entry Index (EX) into the Entry Table, and byte four of the located Entry Table Entry (ETE) is then checked for zero. If the caller is in supervisor state, the PCN translation process is now complete. If the caller is in problem state, the Entry Key Mask in the ETE is checked to see if he is authorized to invoke that particular program call function. If so, translation is complete. (If not, a Privileged Operation exception is generated.)

Next, the routine handles address space switching requirements. If the destination ASID field in the ETE is non-zero, switching is required and the ASID is used as input to the ASN translation process to locate the proper ASN Second Table Entry (ASTE) that contains all the new address space's control information. (Paragraph 3.2.10 describes the ASN translation process.) When the ASTE is returned, the following status changes are made:

- The current primary address space is made the secondary.
- The Linkage Table Designator address (CR5) is set from the ASTE.
- The new Authorization Index (AX) is set from the ASTE.
- If required, a Space Switch Event is noted for later processing.
- The new STO is loaded into CR1.
- The old primary ASID is put into general register 3.

If space switching is not required, the primary address space is also made the secondary, and the old primary ASID is put into general register 3.

At this point, any address space switching and/or housekeeping that is required has been done, and all that is left is program linkage. General register 14 is loaded with the address of the next instruction after the Program Call (the program old PSW instruction address), and the low bit of the register is set to correspond to the PSW problem state bit. The old Program Key Mask (PKM) is OR'd with the Entry Key Mask from the ETE to create the new PKM. General register 4 is loaded with the latent parm word from the ETE. The resume PSW instruction address is then set to the entry point address of the invoked routine. If the new routine is to run in problem state (ETE byte 7, bit 7 is 1), the problem bit is set on in the resume PSW. At this point the actual program call processing is done. If the MVS system trace is disabled, a check is made for the Space Switch Event required flag. If on, an SSE is simulated. If not, exit is made to the new routine via the resume PSW. If tracing is enabled, a trace table entry is made in the same manner as for the EF trace instructions, as described in paragraph 3.1.8. (The format of the entry is given in figure 3-1). Processing then continues as if trace was disabled.

Chapter 4 contains a flowchart of this process.

3.2.2 Set Address Space Control Simulator - SPAB219A

If the caller has translate on, the instruction operand is decoded, the base register (if any) retrieved and the displacement (if any) added in. If bits 20-23 of the resulting address are zero, the request is for primary mode, and the XM mode flag (PSAXMODE) is zeroed to set the mode, the primary STO is loaded to CR1, and control returns to the caller.

If bits 20-23 are not zero, bits 20-22 are checked to ensure that they are zero. If so, the request is a valid request for secondary mode, the XM mode flag is set to X'80' (secondary mode), and the secondary STO is loaded to CR1. Control then returns to the caller. If bits 20-22 are not zero, a specification exception is simulated.

3.2.3 Insert Virtual Storage Key Simulator - SPAB223A

After setting the recursion indicator (PSAPCFB4) and ensuring the caller has translate on, the instruction second operand is decoded and the resulting virtual address is tested for validity. If a valid translation does not exist, the page is referenced and the ensuing program check is reflected to the program FLIH for resolution.

If the address has a valid translation, the real address from the | translation is checked to see if it is over the 16Mb boundary, meaning Extended Addressing is active. If so, bit 4 of the PSW is turned on. An ISK is then done to get the storage key, and the first operand register is decoded and retrieved. If bit 4 of the PSW was turned on, it is now turned off. The retrieved operand value then has the retrieved key OR'd into it, and the result is stored back into the register or save area, as required. Exit is then made to the caller.

3.2.4 Insert Address Space Control Simulator - SPAB224A

If the caller has translate on, the instruction operand is decoded and the register contents retrieved. The resume PSW condition code is set to zero, and the result byte from the instruction (byte 2 of the operand register) is also zeroed. If the CPU is in primary mode, the register is saved with the zero byte, and control returns to the caller. If the CPU is in secondary mode, the result byte is set to X'01' and condition code 1 is set before control is returned.

3.2.5 Set Secondary ASN Simulator - SPAB225A

If the caller has translate on, the instruction operand register is decoded and retrieved, and the destination ASN is compared to the current primary ASN. If they are the same, and ASN translation is allowed, the secondary ASN and STO are set to the values of the primary.

If the destination ASN is not the same as the current primary, then the new ASN must be translated to obtain the necessary control information. Dat is turned off, and the ASN translate routine is called to ensure that the target ASN is a valid secondary address space for this user. (See paragraph 3.2.10 for an explanation of ASN translation and authorization checking.) If the ASN has a valid translation, the new secondary STO and ASN are copied from the retrieved ASTE and set into the proper control register images. If the CPU is in secondary mode, the new secondary STO is loaded to real control register 1.

If tracing is enabled, a trace table entry is made in the same manner as for the EF trace instructions, as described in paragraph 3.1.8. (The format of the entry is given in figure 3-1). After the trace table entry is created, or if trace is not active, control returns to the caller.

Chapter 4 contains a flowchart of this process.

3.2.6 Extract Primary/Secondary ASN - SPAB226A/SPAB227A

If the caller has translate on, and if the EPAR/ESAR is not the subject of an execute, the instruction is overlaid with a Load Halfword (LH) (since the operation of EPAR/ESAR in this case is identical to that of LH). The target register of the LH is that of the EPAR/ESAR and the displacement is that of the primary (EPAR) or secondary (ESAR) ASN in the PSA simulated control registers. After the patch has been built, DYNAMFIX is called to place the patch, and exit is made to the caller.

If the instruction is EXECUTEd, it cannot be overlaid and must be decoded and simulated each time it is encountered. The destination register is decoded, the proper ASN set into it, and return made to the caller.

3.2.7 Program Transfer Simulator - SPAB228A

If the caller has translate on and is not in secondary mode, the instruction operand is decoded, with a fast path if the operand is 'R3,R14', the MVS convention. The state requested by the issuer in the low bit of the second operand is checked to ensure the caller has not requested a transfer to supervisor state from problem state.

The simulation next handles address space switching. If the destination ASID field in the first operand (low halfword) is the same as the current primary, space switching is not required, and the resume PSW instruction address is set from the second operand, secondary ASN and STO are set to those of the primary, and the proper state (supervisor or problem) is set. Trace and Space Switch Events are then checked.

If space switching is required, the destination ASID is used to locate the proper ASN Second Table Entry (ASTE) that contains all the new address space's control information. Authorization checking is done to ensure that this caller is authorized to use the requested address space as a primary address space. Paragraph 3.2.10 describes the ASN translation and authorization process. When the ASTE is returned, the following status changes are made:

- The Linkage Table Designator address (CR5) is set from the ASTE.
- The new Authorization Index (AX) is set from the ASTE.
- If required, a Space Switch Event is noted for later processing.
- The new STO is loaded into CR1 and set into simulated CR1 and CR7.
- Supervisor or problem state is set as requested.
- The primary and secondary ASN are set to the requested new ASN.
- The PKM is reset to what it was on entry to the PC.
- The resume PSW instruction address is set from the second operand.

If a Space Switch Event is required, a flag is set for later inspection.

If the MVS system trace is disabled, a check is made for the 'Space Switch Event Required' flag. If on, an SSE is simulated. If not, exit is made to the new routine via the resume PSW. If tracing is enabled, a trace table entry is made in the same manner as for the EF trace instructions, as described in paragraph 3.1.8. (The format of the entry is given in table 3-1.)

Chapter 4 contains a flowchart of this process.

3.2.8 Move to Primary and Move to Secondary Simulations - MVCXCOMM

Routine MVCXCOMM receives control when an MVCP or MVCS opcode (X'D9' or X'DA') is encountered. If the caller has translate on, the instruction operands are decoded, the base registers (if any) are retrieved and the displacements added in. (The length register is decoded, retrieved, and saved into a work area for use in setting the condition code at the end of the simulation.) If the length was zero, the move is a NOP and control returns to the caller with condition code zero. If the length is greater than 256, it is set to 256, then decremented by one for working purposes. The key register, used for access checking for the secondary space operand, is then decoded and retrieved.

At this point, the source and sink data operands must be checked for accessability. There must be a valid translation for the addresses, and the storage keys must agree with the PSW key or secondary key, as required. There are four routines that do the checking, depending on the instruction being simulated and whether the CPU is in primary or secondary mode. The simulation is optimized for MVCS in primary mode, by far the most common simulation. The checking involves testing both ends of the operand for access, using the proper keys and segment table. The routines that do the checking, and the origin of the keys and STOs are:

- MVCP, primary mode - after label MVCPPORS
  - Operand 1 - PSW key, Active STO
  - Operand 2 - R3 key, STO from CR7
- MVCP, secondary mode - Label MVCPS
  - Operand 1 - PSW key, STO from CR1
  - Operand 2 - R3 key, Active STO
- MVCS, primary mode - (No Label, fall through)
  - Operand 1 - R3 key, Active STO
  - Operand 2 - PSW key, STO from CR7
- MVCS, secondary mode - Label MVCSS
  - Operand 1 - R3 key, STO from CR1
  - Operand 2 - PSW key, Active STO

If there is no valid translation for part of the data, or the keys do not match, a program check occurs that will be handled by IEAVEXM2.

If the caller is in problem state, and is not allowed to use the secondary space key, a protection exception is generated. If the user is authorized, or in supervisor state, DAT is turned off and the operands are checked for page crossing. If the move is one byte the page crosser checking is skipped.

After page crossing checks, there are extended addressing checks. If extended addressing is active on the system (determined from the inspection of PSAHWFB) and either of the operand addresses is above the 16Mb boundary, bit 4 of the PSW is turned on to allow the accessing of the high storage.

The move is then simulated by either an EXECUTEd MVC or an IC/STC sequence (for one byte moves). Condition code 3 is set if the original length was greater then 256; otherwise, condition code zero is set. Control then passes to the resume exit.

If a page crosser has been detected, further processing depends on which operand(s) crossed. If only one crossed, the move is done with two EXECUTEd MVCs. If both operands crossed, three moves are required. In all cases, the addresses involved are checked for residence above the 16Mb boundary, and PSW bit 4 turned on before the moves if a high address has been found.

3.2.9 Move With Key Simulation

The initial part of the MVCK simulation is common with that for the MVCP and MVCS simulations up to the point that the operands are decoded and retrieved. After the key register has been retrieved, the simulation becomes unique to MVCK. The MVCK instruction uses only one segement table, so the checking with different segment tables for access is not required. Also, DAT is not required to be on. Since page crossing checks are not required, it is only necessary to access the first and last byte of each operand in the proper key. (A check for page crossing is not needed because moves are done DAT on or DAT off, depending on the caller's mode, so all storage will be logically contiguous. The other XF moves are always done DAT off, so this will not usually be the case for them.) The first operand key is checked with the PSW key, and the second with the R3 key. If one of the keys is improper, or if DAT is on and access to of one of the operands causes a page fault, the resulting program check is reflected to the original instruction by IEAVEXM2.

If the user is in problem state, the R3 key is checked against the PSW key mask. When all checks have been successfully passed, the move is done with an EXECUTEd MVC, the proper condition code is set (depending on the length register), and control returns to the caller via the standard resume exit.

3.2.10 ASN Translation and Authorization - ASNTRAN

The ASN translation routine is called by the Program Call, Program Transfer and Set Secondary ASN simulation routines when a space switch is required. The ASNTRAN routine takes as input an ASN (ASID), and using the ASN translation tables built during IPL, returns an ASTE that contains information about the target address space. This information includes such things as the segment table origin address and the LTD, AX and Authority Table Origin.

The ASN translation process is similar to that of a virtual to real address translation. The index into the ASN First Table (AFT) is the high ten bits of the ASN, called the AFX. The AFX is multiplied by 4 to provide the offset into the AFT to locate the AFTE. The start of the AFT, the AFTO, is contained in CR14 bts 12-31, and is the same for all users of the system. If bit 12 is zero, translation is not allowed, and a special operation exception is generated.

The valid/invalid bit of the AFTE is first inspected, and then the AFTE itself is format checked. If it passes these tests, the AFTE is used as the start of the AST (ASN Second Table). The low-order 6 bits of the ASN are multiplied by 16 to get the index into the second table (ASX), and are added to the ASTO obtained from the AFTE. The result is a pointer to the ASTE, the ASN Second Table Entry. The ASTE is then format checked, and if the format is valid and the invalid bit is not on, the translation process is complete.

If the translation was performed for the Program Call simulation, the process is complete and control returns to the simulation routine. If the call was from PT or SSAR (the caller is identified from the value in PSAPCFB4), ASN authorization checking is required. This process ensures that the issuer of the PT or SSAR is authorized to request the address space he has specified in the instruction as either a primary or secondary space, respectively.

ASN authorization involves locating the Authorization Table Origin (ATO) from the ASTE, and indexing in using the caller's AX to do the actual check. Each byte in the AT (Authorization Table) contains information for four AX's. The AX is divided by 4 to locate the proper byte, and the proper pair of bits is located from the last two bits of the AX. For primary authorization, the first of the pair must be one; for secondary the second bit must be one. The proper bit is calculated and a TM instruction is EXECUTEd to perform the check. If the check passes, control returns to the appropriate simulation routine. Otherwise, an authorization exception is generated.

Chapter 4 contains a flowchart of this process.

3.3 EXTENDED ADDRESSING (EA) SUPPORTED INSTRUCTIONS

Extended Addressing on the Amdahl 470 rests on three assumptions:

1. Storage key instructions operate only in 4K mode.

2. Turning on bit 4 of the PSW enables 31-bit real addressing.

3. The Extended Memory hardware feature is installed.

MVS/SPA expects all three conditions to be met. (A modification to IEAVNIP0 ensures that the appropriate hardware features are enabled if present.)

3.3.1 Insert Storage Key Extended Simulator - SPAB229A

First, the instruction operand registers are decoded and their contents retrieved. Then bit 4 of the PSW is turned on if required, an ISK is done to get the storage key, and PSW bit 4 is turned off. The retrieved first operand is used as the first operand of the ISK, and the second operand provides the address to use. After the ISK, the first operand register is stored back into the proper location, and control the returns to the caller from the resume exit.

3.3.2 Reset Reference Bit Extended Simulator - SPAB22AA

First, the instruction operand register is decoded and its contents retrieved. Then bit 4 of the PSW is turned on if required, and an RRB is done to set the condition code and reset the reference and change bits. The resulting condition code is then set into the resume PSW, and control passes back to the caller from the resume exit.

3.3.3 Set Storage Key Extended Simulator - SPAB22BA

First, the instruction operand registers are decoded and their contents retrieved. Then bit 4 of the PSW is turned on if required, an SSK is done to set the storage key, and PSW bit 4 turned off. Control then returns to the caller from the resume exit.

3.4 IPK/SPKA IN PROBLEM STATE

Part of the implementation of the DAS feature is the changing of the two PSW key handling instructions, Insert PSW Key and Set PSW Key From Address, from privileged instructions to 'semi-privileged'. With appropriate authorization, any problem program is now allowed to use these instructions. A CPU without the DAS feature will not support this execution, and causes a privileged operation exception (interrupt code X'02'). This exception is detected and reflected to IEAVEXMS to handle. Routine CHECKPI2 is given control when a privileged operation exception is detected. It then chooses the proper simulation routine to receive control.

3.4.1 Insert PSW Key Simulator

The IPK simulator sets the key from the program old PSW into the low byte of register 2, and then ANDs out the low nibble. No authorization checking is done, as the only requirement is extraction authority. Control is then returned to the caller via the standard resume exit.

3.4.2 Set PSW Key From Address Simulator

The SPKA simulator decodes the operand register, and if non-zero, retrieves its contents and adds in the displacement to obtain the requested new PSW key. The bit in the PSW key mask corresponding to the new key is checked. If it is one, the new key is placed into the resume PSW key field, and control returns to the user via the resume exit. If the PSW key mask bit is zero, the original privileged operation exception is reflected back to normal RTM processing through the error exit.

3.5 EXIT PROCESSING

The simulation routines and the recursion routine will exit either directly back to the caller, in the case of a successful simulation, or to the program check FLIH in the event of an error detected during format checking, operand validation in a simulation, a Space Switch Event or a PER event. Due to MVS/SPA being imbedded in the program check handler, some housekeeping must be done to avoid problems in either case.

3.5.1 Resume Exit

The resume exit is used after a successful simulation to return control to the instruction stream as directed by the simulation. The point of return is either the next sequential instruction after the one being simulated, or another location. The resume exit will first clear out the recursion indicator byte, PSAPCFB4, to ensure that IEAVEXM2 will not receive control for program checks not related to SPA processing. If the program check that initiated the simulation was not a recursion, the FRR stack is reset to what it was on entry. (The program FLIH had set it to the program check handler stack.) The PI super bit is then reset to avoid the next program check being considered a recursion. If one of the other bits was on in the program check status word (PSAPCFUN), the SUPER bit and FRR stack are not reset. (An example of this would be an occurrence of an EF 'Trace Program Interrupt' instruction.) Checking is now done to see if a PER event was reported with this interrupt. If so, the rest of the interrupt code is zeroed out, and the error exit routine is called at label EROREXT3. If no PER interrupt was indicated, the SPA control flag is reset, as it is only valid during simulation. The caller's registers are then reloaded, and control returns by LPSW of the resume PSW prepared by the simulation routine or low core trap.

3.5.2 Error Exit

The error exit is used when control must be passed on to the program FLIH and/or RTM. Some of these cases are:

- A simulation routine has detected an error in an operand.

- An instruction has been issued in an improper environment.
- An interrupt was passed to IEAVEXMS that it can not process.
- A page fault has been encountered accessing an operand.
- A Space Switch Event is to be signaled.
- A PER event has occurred.

There are five points of entry into the error exit, depending on the processing required. Each falls through to the next.

1.	EROREXIT	PSA protection is disabled.
2.	EROREXT1	The resume PSW is made the program new PSW.
3.	EROREXT2	The PSW address set in the program old PSW is checked to ensure that it is not the address of the work area where the target instruction of an EXECUTE has been built.
4.	EROREXT3	PSA protection is re-enabled.
5.	EROREXT4	The remainder of the exit processing.

The specific entry points are called as required, depending on what part of the exit processing has already been done by the simulation routine or is already correct.

The remainder of the exit processing is primarily MVS housekeeping and clean-up. The SPA control flag and recursion bytes are cleared, and the rest of the program check recursion word (PSAPCFUN) is checked to see if the program check that resulted in the entry to IEAVEXMS was a true recursion. If not, the program interrupt code being passed on is checked for an operation exception or a privileged operation exception. If the interrupt is not to be either of these, the previously current FRR stack is restored to the current, and the program interrupt handler SUPER bit is turned off. The caller's registers then are restored and the program new PSW loaded, entering normal program check processing. If the interrupt being passed back is one of the two, a special entry point in the program check FLIH (IEAVEPC1) is used to ensure that the interrupt is not routed back to IEAVEXMS in a loop. The program FLIH base address is set into register 9, (entry IEAVEPCB), the special entry point address into register 11, the 'LCCA not validated' bit is set (program FLIH housekeeping), the user's registers are copied to the program FLIH register save area, DAT is turned off if it is on, and the FLIH is branched to.

If the interrupt being passed back is from an instruction in the program check handler itself, the recursion return point must be used. Recursion exit processing is identical to that of the non-recursion processing, save that if the interrupt to be passed is a 1 or 2, the entry point IEAVEPC2 is used.

3.5.3 Setting Exceptions

When a simulation exit detects a format, environment or other error, the program interrupt to be generated is usually not the same as the current interrupt code indicated. In that case, the simulation branches to an exception creation routine that sets up the proper interrupt code and instruction length code.

The exception creation routines restore registers 12-15 if requested and then disable PSA low storage protection. The new interrupt code is then set, including the PER bit if it was on in the original interrupt. The ILC is set to either the length of the instruction or zero, depending on whether the definition of the new interrupt causes suppression or nullification, respectively. The PSW is also backed up if required to agree with the ILC. In some cases a translation exception address is set. Control then passes to the normal error exit at label EROREXT1.

3.6 SPA RECURSION ROUTINE - IEAVEXM2

Entry point IEAVEXM2 is called directly from the program FLIH when a program check occurs and a non-zero value has been set into field PSAPCFB4. (PSAPCFB4 is the one-byte field that is used as an indicator of which simulation routine is running.) The various values that can be put in this field are given in table 3-2.

IEAVEXM2 first checks to see if a TPROT instruction was being simulated; if so, the proper condition code is set into the resume PSW to reflect the cause of the interruption: a page or segment fault or a protection exception. If another type of exception occurred, the exception is reflected back to the program check handler through the normal error reflection exit (see paragraph 3.5.2).

If a TPROT was not being simulated, a check is made first for a segment or page fault. If one has occurred, the translation exception address (TEA) is set to reflect the address space the fault belonged to, then exclusive OR'd with the PSAXMFLG value. The XOR is done because IEAVEXM2 sets the proper primary or secondary address space bit. (The program FLIH XOR's in the secondary mode bit setting only.) After the fault has been resolved, the PSW is backed up to point to the instruction being simulated that 'caused' the fault to be re-dispatched and re-simulated.

Depending on the mode flag (PSAXMFLG), either the primary or secondary STO is loaded, and the program interrupt SUPER flag (bit PSAPI in PSASUP1) is reset if it is on. If MVCP or MVCS was executing, registers 12-15 are reloaded from the save area. At last SPA calls the program FLIH to handle the interrupt from the simulation.

3.7 RESERVING PSA PATCH AREA

MVS/SPA assumes that it is the only user of the PSA patch area (PSA free area); any product that also uses the PSA patch area will cause conflicts with MVS/SPA. To resolve the conflict, portions of the PSA area can be set aside from SPA use.

The areas of PSA used are determined by the values of three pairs of default and user-adjustable boundary constants; by superzapping the user constants, the user can change or move the PSA patch areas. They are labeled 'SEPxxxxx', where xxxxx defines the use of the constant, and are located near the end of the module. (The defaults are labeled 'DEFxxxxx' where xxxxx is the same as for the user constants.)

There are two constants for each area, an upper and a lower bound. If only one bound is changed, the default will be used for the other. To disable the use of an entire area, change the lower bound constant for that area to X'FFFF'.

The PSA allocation routine PSAALLOC examines these pairs the first time a PSA free area must be used. If no changes are specified, the area is not totally disabled, or the replacement parameters are in error, the defaults will be used. The low address of the first two parameters is rounded up to an 8-byte boundary and the low address of the third area is rounded up to a 32-byte boundary. The following conditions are checked:

- The low address of a pair not in a free area.
- High address of a pair being outside of the PSA.
- High address of a pair lower than the low address.
- The high and low address of a pair not in the same free area.
- Any area not big enough for two maximum size traps.
- Any of the areas overlap.

There are currently three free areas available to MVS/SPA:

1.	PSAUSEND to PSAUS2ST	X'6C8' to X'800'	X'138' bytes
2.	PSAUS2ND to PSAPFXA	X'810' to X'AE0'	X'2D0' bytes

3. PSASTAK+768 to PSAAMDST X'F00' to X'FE0' X'E0' bytes

(Labels PSAPFXA and PSAAMDST are defined in the Amdahl PSA mapping macro extension SPAPSA.)

| MVS/SPA uses fixed areas from X'AE0' to X'AFF' and X'FE0' to X'FFF'. These areas cannot be moved without a reassembly of the product.

Please note that reducing the amount of available PSA may cause some performance degradation.

3.8 SUPPORT FOR THE EXECUTE INSTRUCTION

MVS/SPA handles interrupts caused by EF, XF and EA instructions which are targets of an EXECUTE instruction. If such an instruction is identified as the cause of the interrupt, the operands of the EXECUTE are decoded and fetched. A special flag is set to cause routines that would normally patch or trap the instruction to simulate it or not to save a trap. The target instruction of the EXECUTE is moved to a work area, and the modification register, if any, is decoded, fetched and then OR'd into the copy of the instruction skeleton. The address of the EXECUTE is saved so that the PSW address can be set properly should an exception occur. The address of the instruction built in the work area is then used as though it had not been the subject of the EXECUTE, and control is passed back to the start of the opcode decode section.

Should control return to the EXECUTE decode section, the original interrupt will be passed on through to RTM to handle as a normal operation or privileged operation exception.

3.9 PER CONSIDERATIONS

In order to maintain the integrity of the lockwords, the lock handling instructions run as logical extensions of the lock manager. To do this, the lock manager SUPER bit, PSALOCK, is set to ensure that, should a PER interrupt occur, control will immediately be returned to the simulation. (The PER event will be lost.)

Register alteration events may be signaled during the execution of the page fix assist simulation routine. Instruction fetch events for the simulated instruction should always be reported.

Generally, no other PER events will be reported in the simulation of any instruction. Successful branch PER events may be signaled at the original location of an EF instruction, as the original instruction may have been replaced by a branch instruction.

CHAPTER 4 – FLOWCHARTS

The following flowcharts (figures 4-1 through 4-33) are designed to provide a conceptual idea of instruction simulation requirements and processing flows. They do not necessarily coincide with the line-by-line logic of the product, but provide a summary of the processing performed.

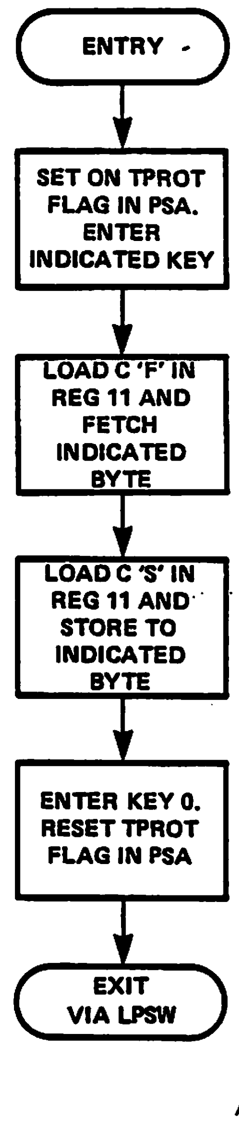
ENTRY
SET ON TPROT FLAG IN PSA. ENTER INDICATED KEY
LOAD C 'F' IN REG 11 AND FETCH INDICATED BYTE
LOAD C 'S' IN REG 11 AND STORE TO INDICATED BYTE
ENTER KEY 0. RESET TPROT FLAG IN PSA
EXIT VIA LPSW
A03113

Figure 4-1. SPAE501 - Test Protect Simulation

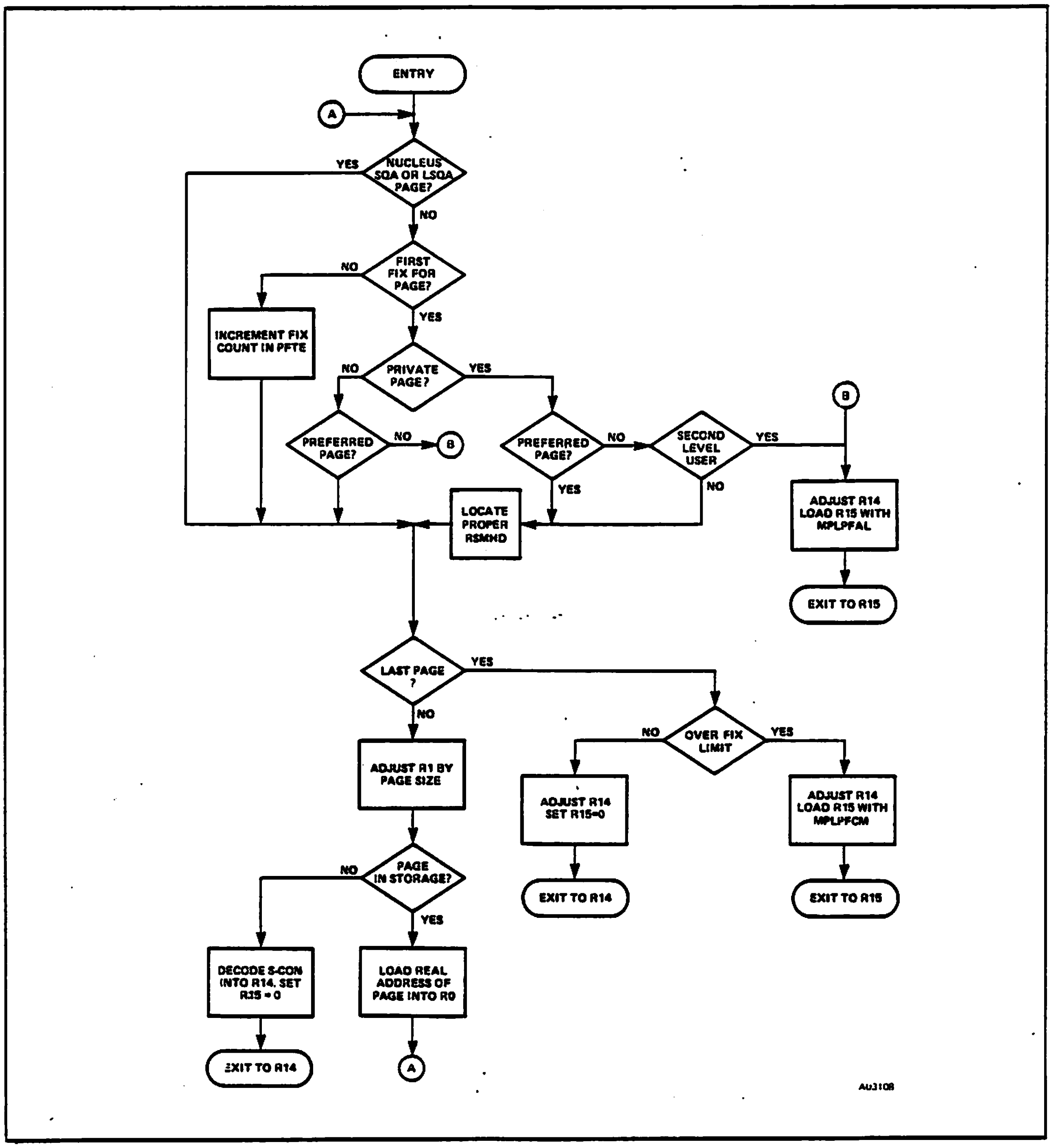

Figure 4-2. SPAE502 - Fix Page Simulation

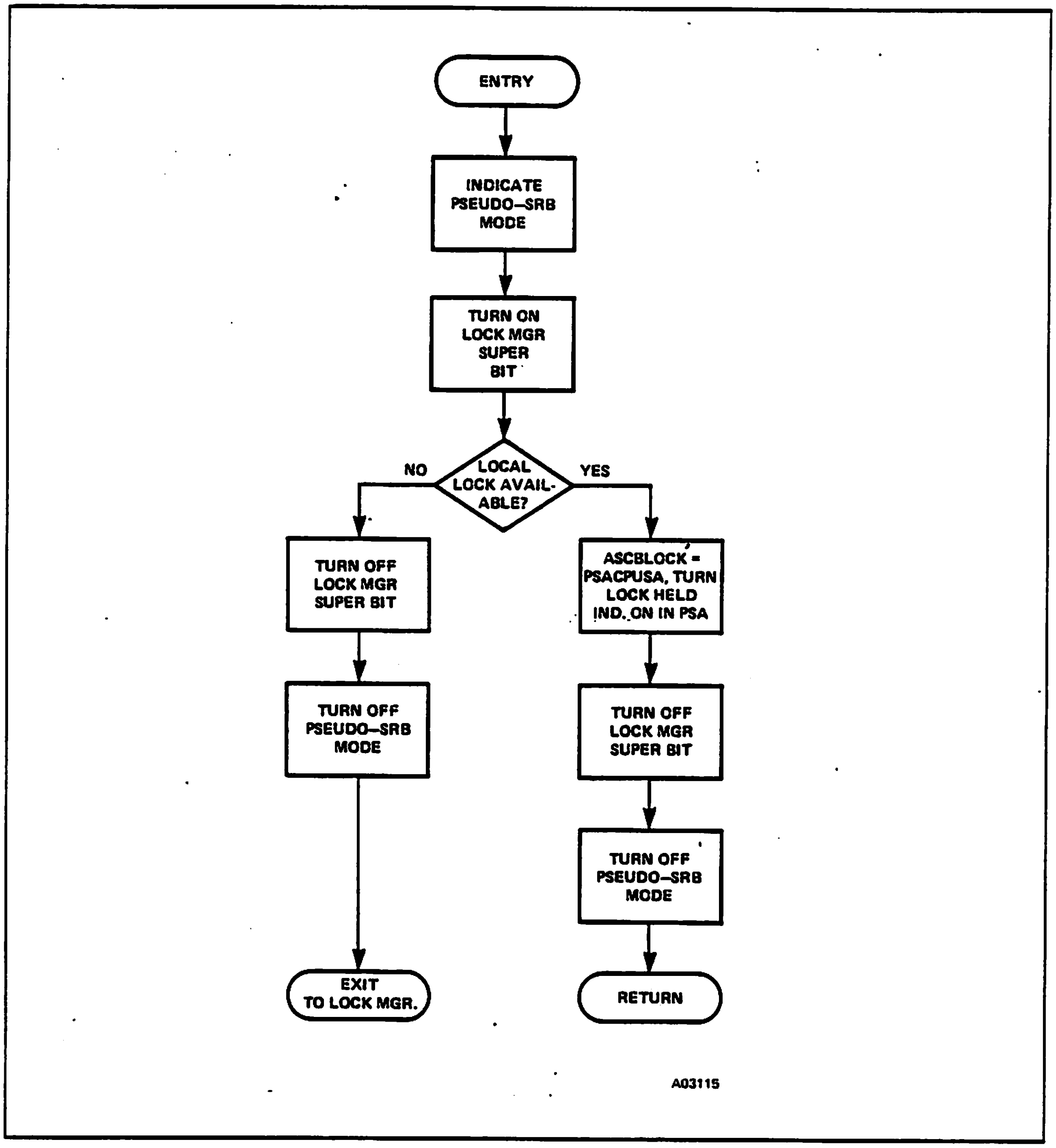

Figure 4-3. SPAE504 - Obtain Local Lock Simulation

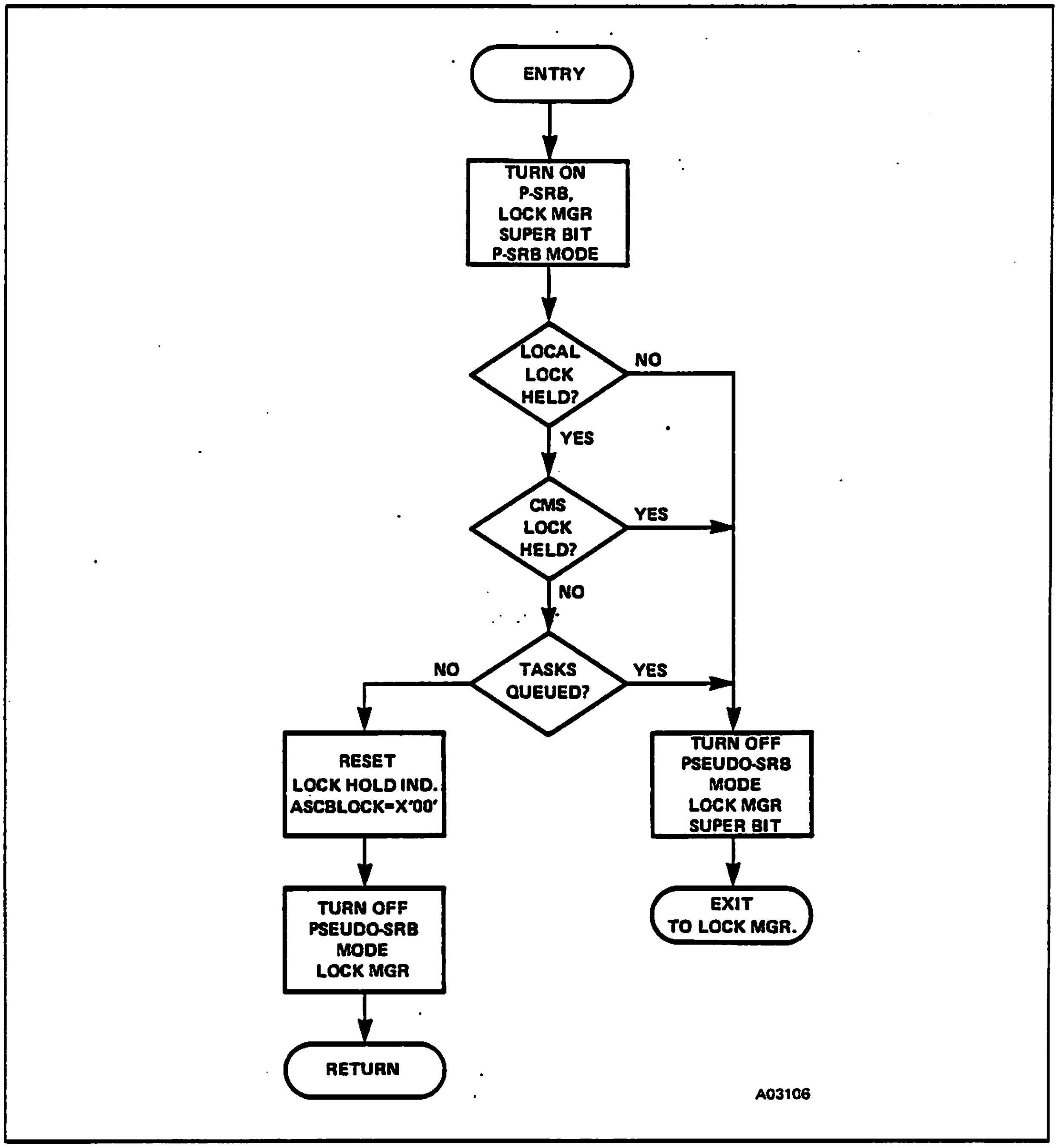
ENTRY
TURN ON P-SRB, LOCK MGR SUPER BIT P-SRB MODE
LOCAL LOCK HELD?
NO
YES
CMS LOCK HELD?
YES
NO
TASKS QUEUED?
NO
YES
RESET LOCK HOLD IND. ASCBLOCK=X'00'
TURN OFF PSEUDO-SRB MODE LOCK MGR
RETURN
TURN OFF PSEUDO-SRB MODE LOCK MGR SUPER BIT
EXIT TO LOCK MGR.
A03106

Figure 4-4. SPAE505 - Release Local Lock Simulation

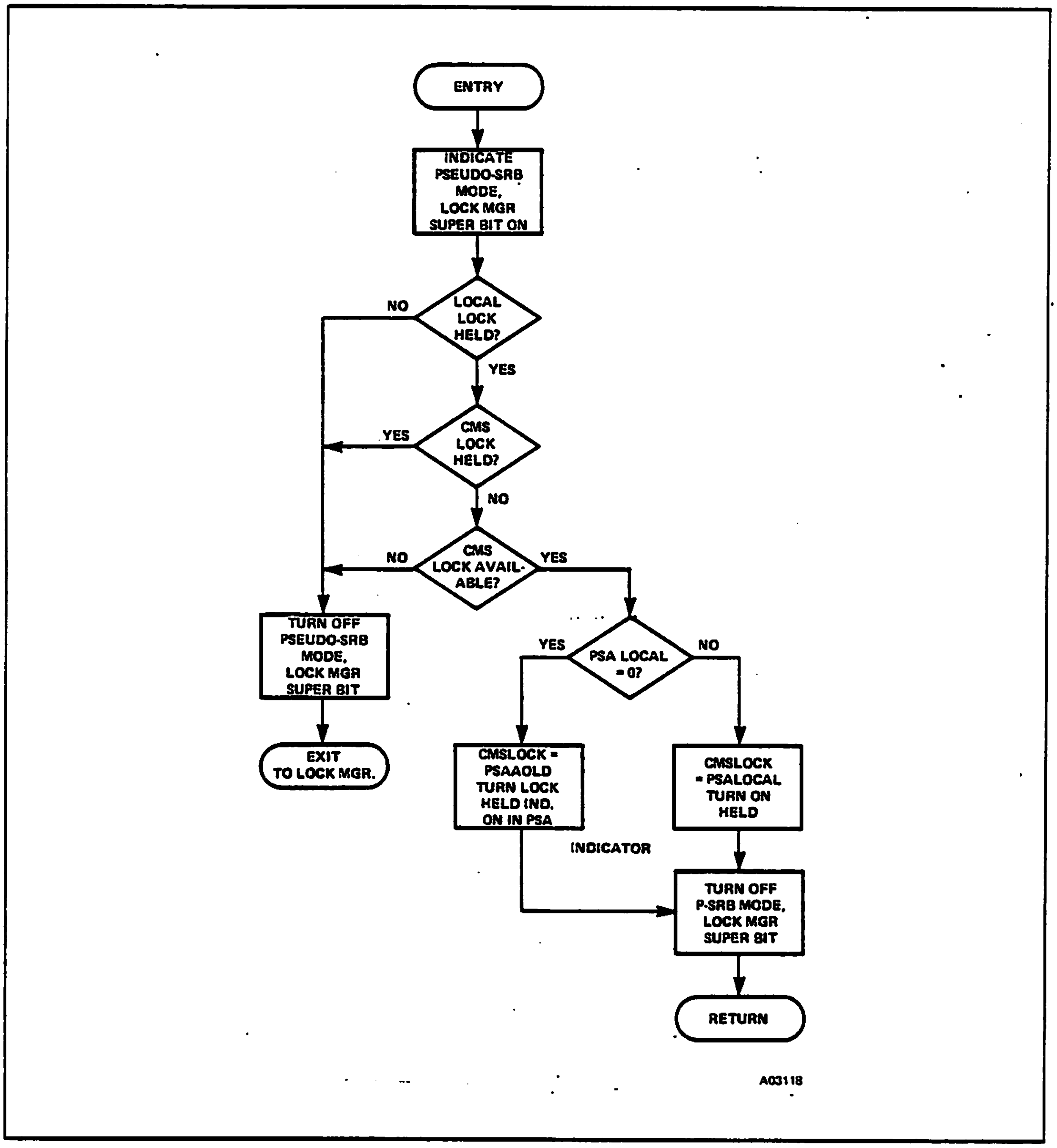
ENTRY
INDICATE PSEUDO-SRB MODE, LOCK MGR SUPER BIT ON
LOCAL LOCK HELD?
NO
YES
CMS LOCK HELD?
YES
NO
CMS LOCK AVAIL-ABLE?
NO
YES
TURN OFF PSEUDO-SRB MODE, LOCK MGR SUPER BIT
EXIT TO LOCK MGR.
PSA LOCAL = 0?
YES
NO
CMSLOCK = PSAAOLD TURN LOCK HELD IND. ON IN PSA
INDICATOR
CMSLOCK = PSALOCAL TURN ON HELD
TURN OFF P-SRB MODE, LOCK MGR SUPER BIT
RETURN
A03118

Figure 4-5. SPAE506 - Obtain CMS Lock Simulation

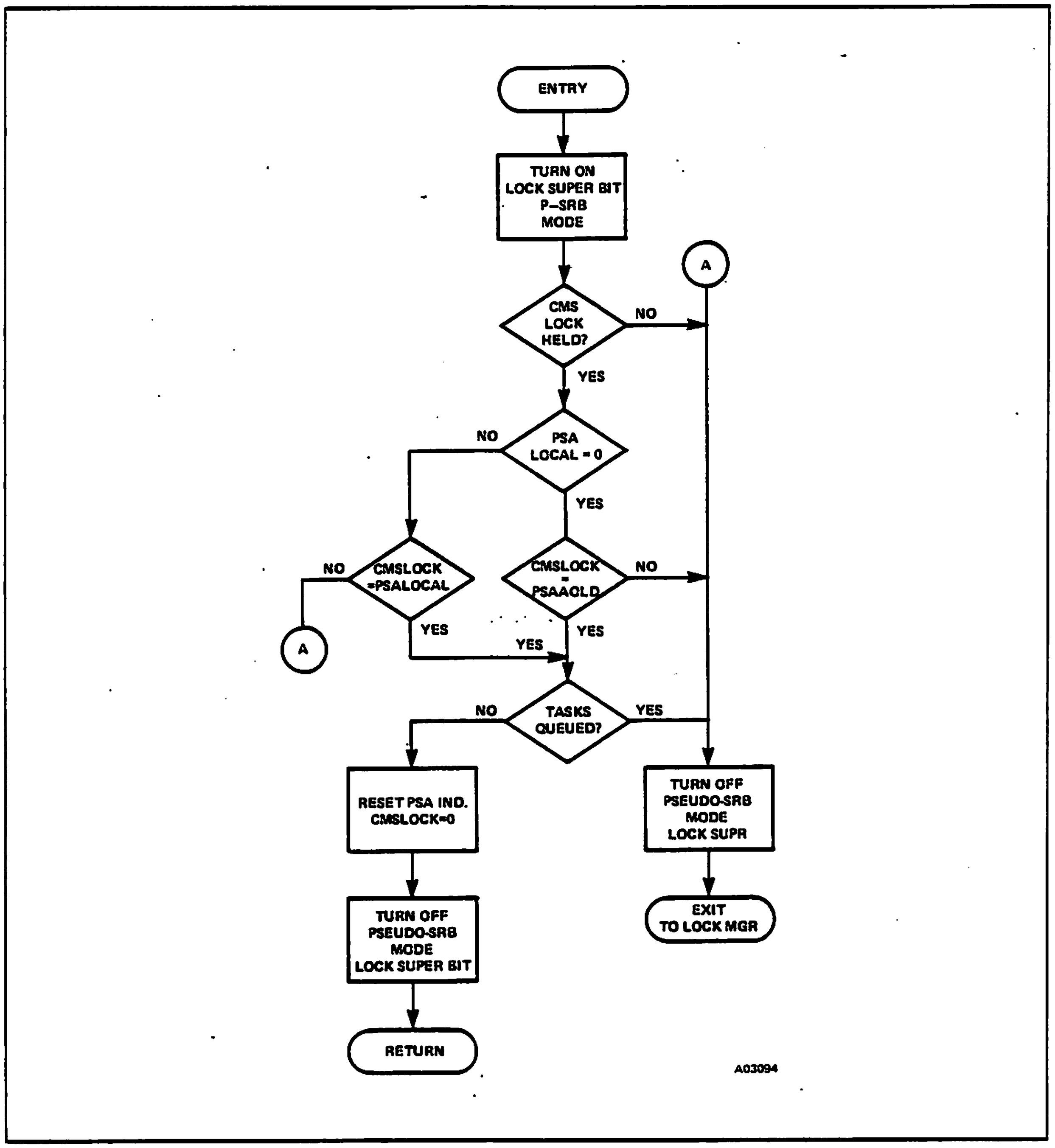

Figure 4-6. SPAE507 - Release CMS Lock Simulation

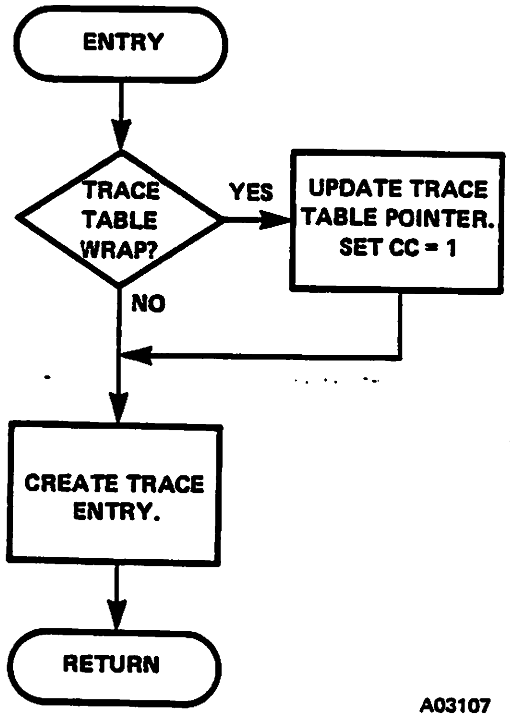

Figure 4-7. SPAE508-D - Trace Instruction Simulation

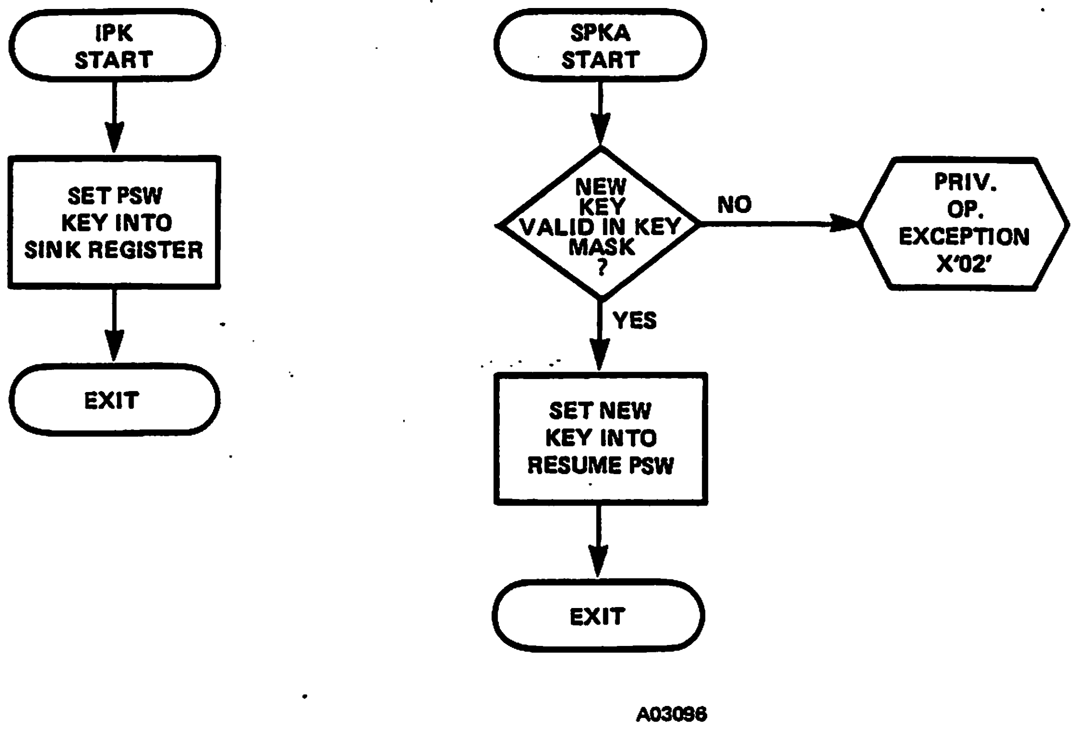
IPK START
SET PSW KEY INTO SINK REGISTER
EXIT
SPKA START
NEW KEY VALID IN KEY MASK ?
NO
PRIV. OP. EXCEPTION X'02'
YES
SET NEW KEY INTO RESUME PSW
EXIT
A03096

Figure 4-8. CHECKPI2 - IPK and SPKA in Problem State

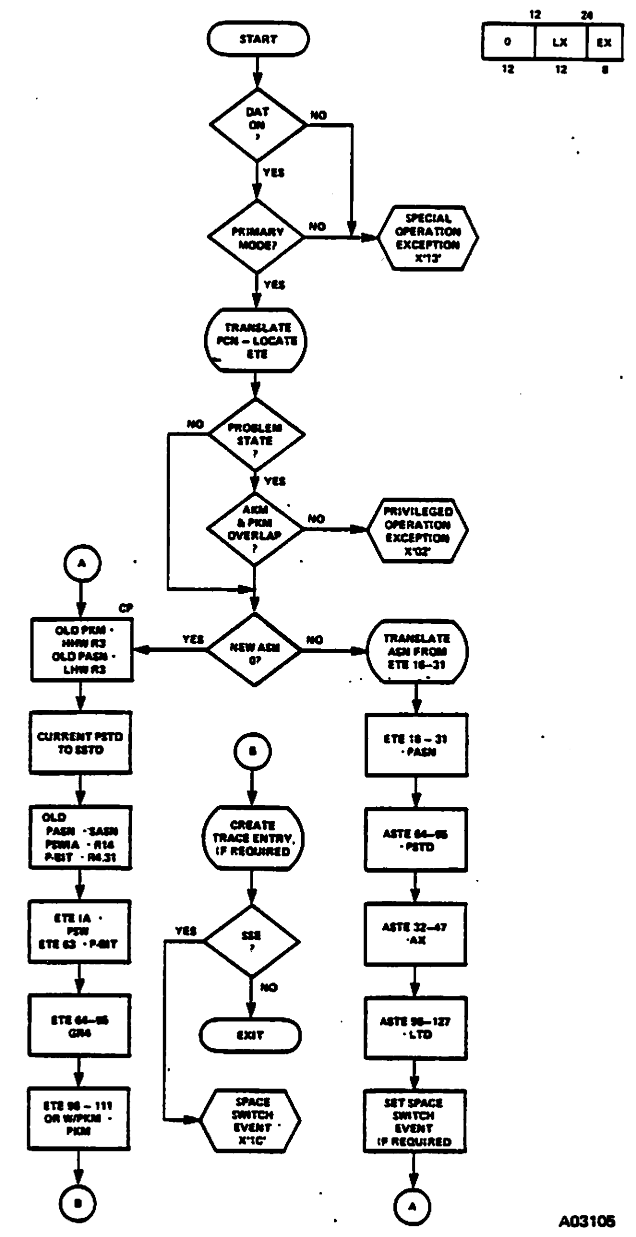
12
24
0
LX
EX
12
12
8
START
DAT ON ?
NO
YES
PRIMARY MODE?
NO
SPECIAL OPERATION EXCEPTION X'13'
YES
TRANSLATE PCN – LOCATE ETE
PROBLEM STATE ?
NO
YES
AKM & PKM OVERLAP ?
NO
PRIVILEGED OPERATION EXCEPTION X'02'
A
CP
OLD PKM · HHW R3 OLD PASN · LHW R3
YES
NEW ASN 0?
NO
TRANSLATE ASN FROM ETE 16–31
CURRENT PSTD TO SSTD
B
ETE 16 – 31 · PASN
OLD PASN · SASN PSWIA · R14 P-BIT · R4.31
CREATE TRACE ENTRY, IF REQUIRED
ASTE 64–95 · PSTD
ETE IA · PSW ETE 63 · P-BIT
YES
SSE ?
NO
ASTE 32–47 · AX
ETE 64–95 GR4
EXIT
ASTE 96–127 · LTD
ETE 96 – 111 OR W/PKM · PKM
SPACE SWITCH EVENT X'1C'
SET SPACE SWITCH EVENT IF REQUIRED
B
A
A03105

Figure 4–9. SPAB218A - Program Call Simulation

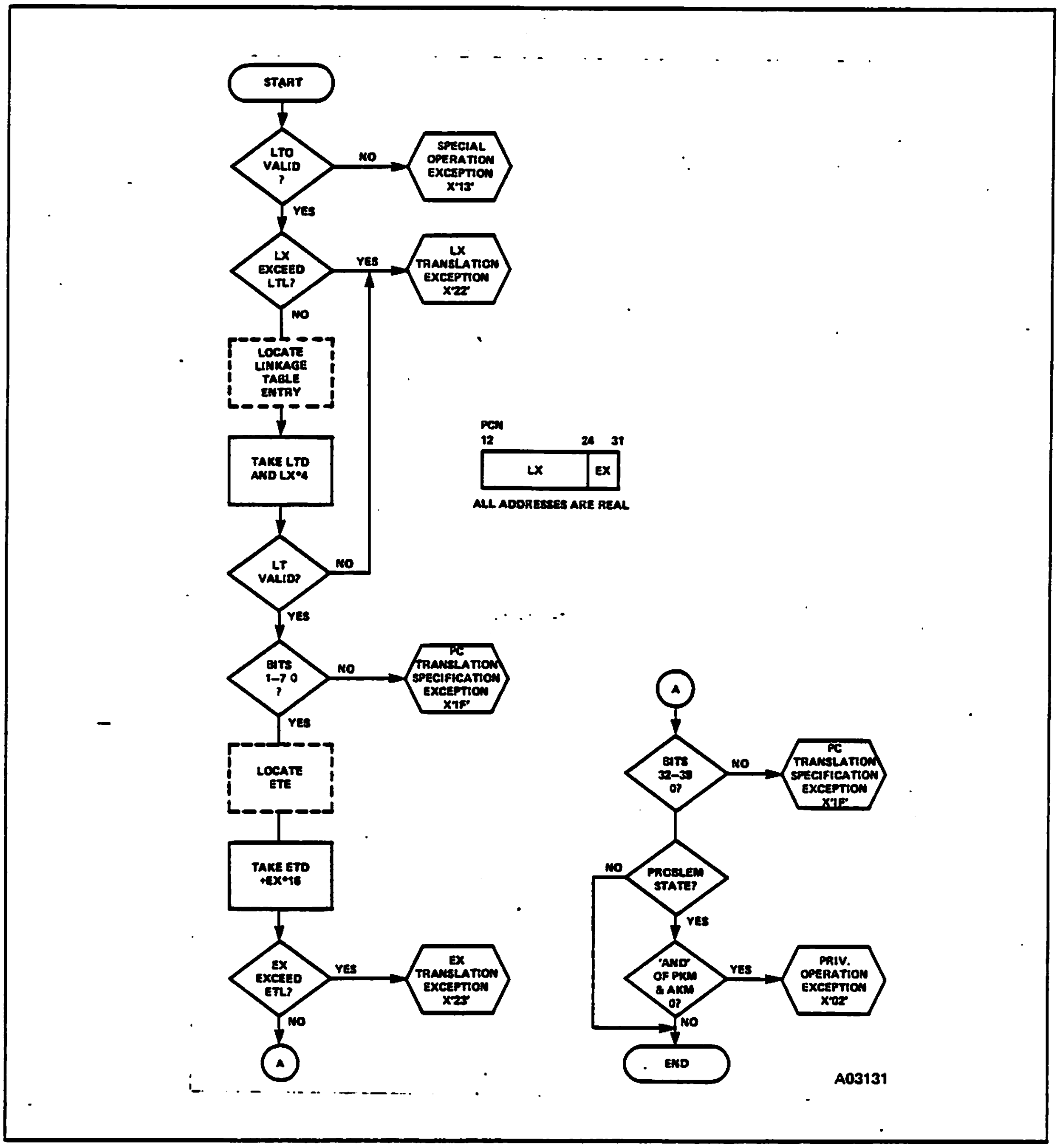
START
LTO VALID ?
NO
SPECIAL OPERATION EXCEPTION X'13'
YES
LX EXCEED LTL?
YES
LX TRANSLATION EXCEPTION X'22'
NO
LOCATE LINKAGE TABLE ENTRY
TAKE LTD AND LX*4
LT VALID?
NO
YES
BITS 1–7 0 ?
NO
PC TRANSLATION SPECIFICATION EXCEPTION X'1F'
YES
LOCATE ETE
TAKE ETD +EX*16
EX EXCEED ETL?
YES
EX TRANSLATION EXCEPTION X'23'
NO
A
PCN
12
24
31
LX
EX
ALL ADDRESSES ARE REAL
A
BITS 32–39 0?
NO
PC TRANSLATION SPECIFICATION EXCEPTION X'1F'
PROBLEM STATE?
NO
YES
'AND' OF PKM & AKM 0?
YES
PRIV. OPERATION EXCEPTION X'02'
NO
END
A03131

Figure 4–10. Program Call Number Translation

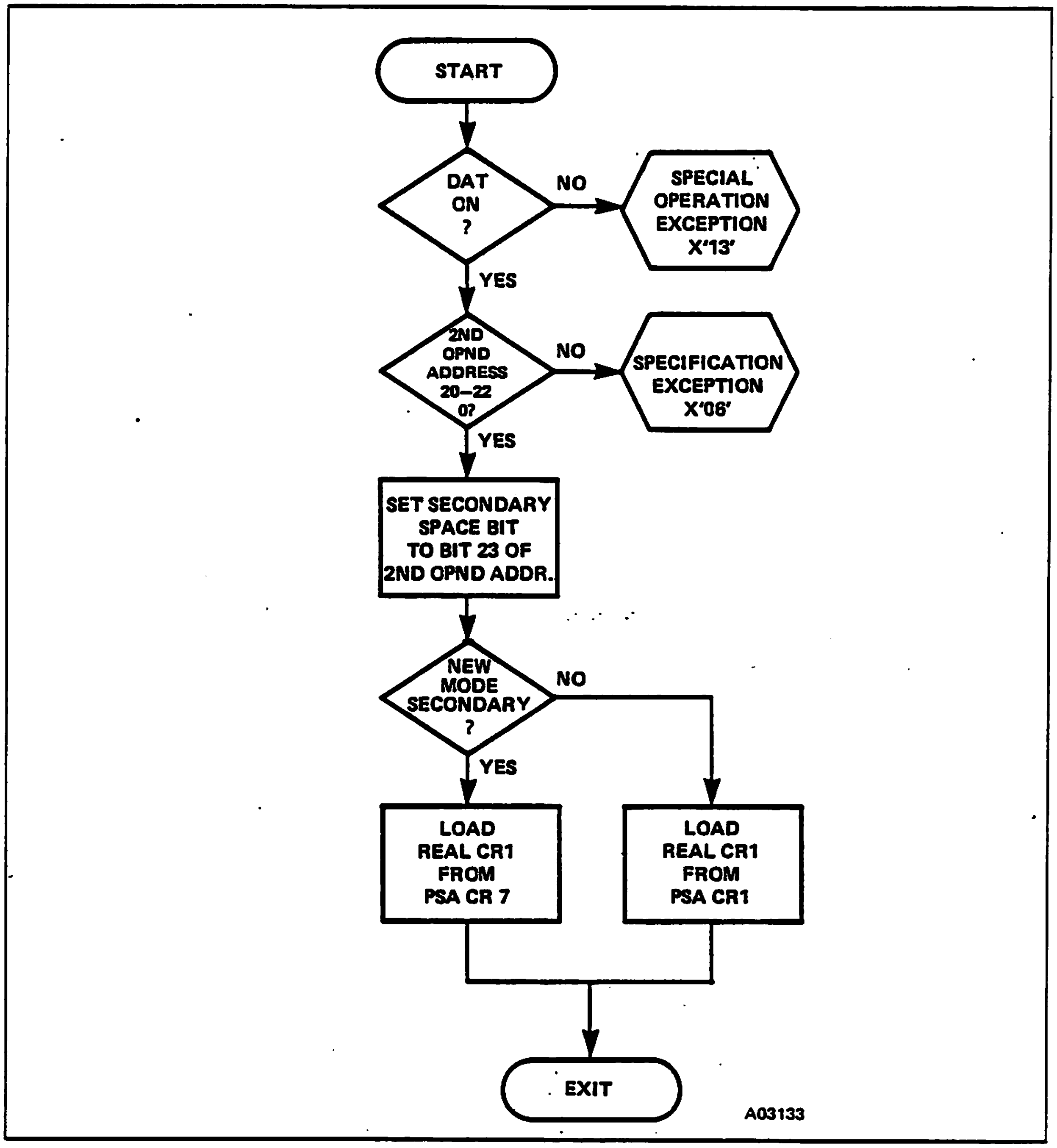
START
DAT ON ?
NO
SPECIAL OPERATION EXCEPTION X'13'
YES
2ND OPND ADDRESS 20–22 0?
NO
SPECIFICATION EXCEPTION X'06'
YES
SET SECONDARY SPACE BIT TO BIT 23 OF 2ND OPND ADDR.
NEW MODE SECONDARY ?
NO
YES
LOAD REAL CR1 FROM PSA CR 7
LOAD REAL CR1 FROM PSA CR1
EXIT
A03133

Figure 4-11. SPAB219A - Set Address Space Control Simulation

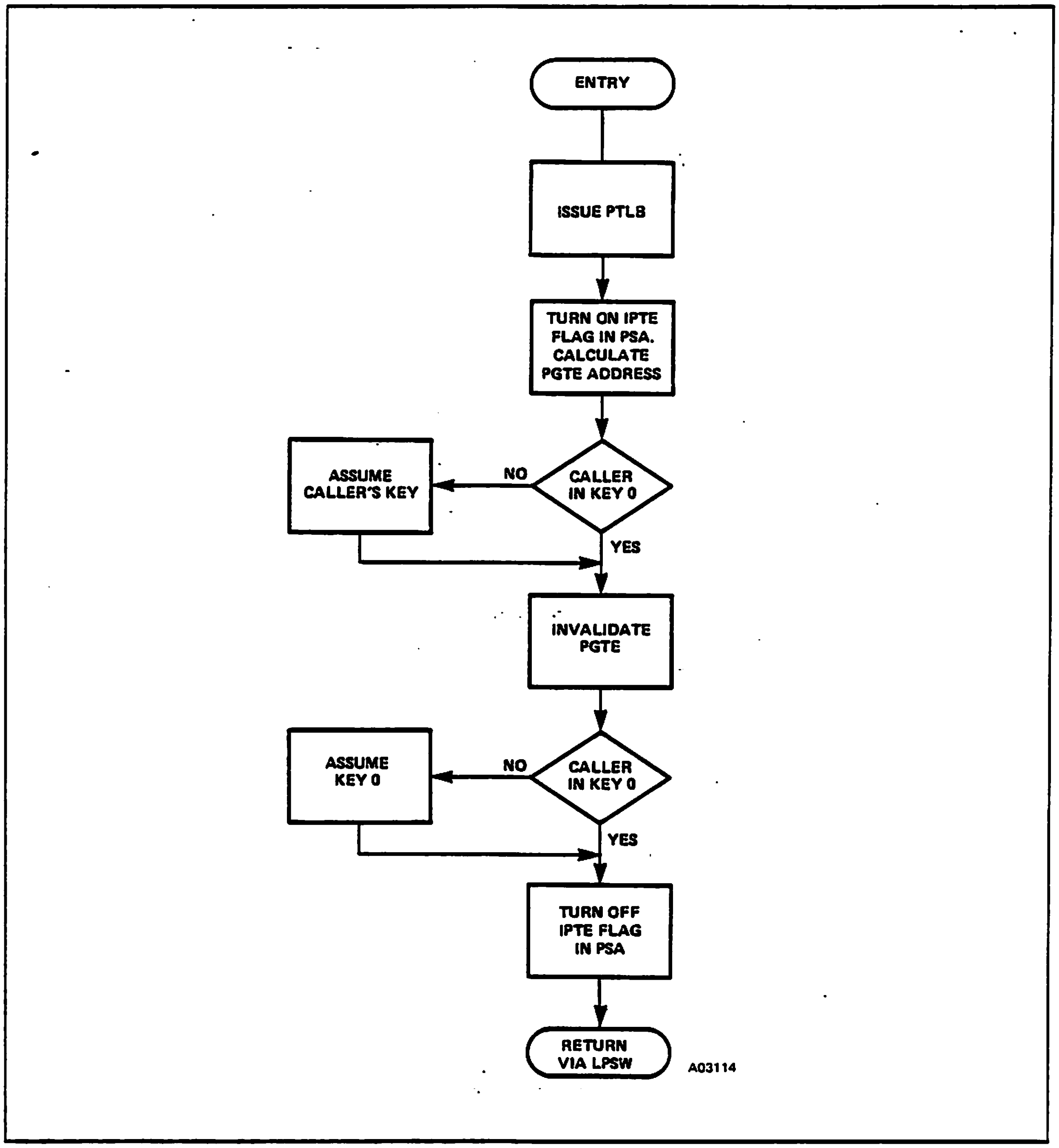

Figure 4-12. SPAB221A - IPTE Simulation

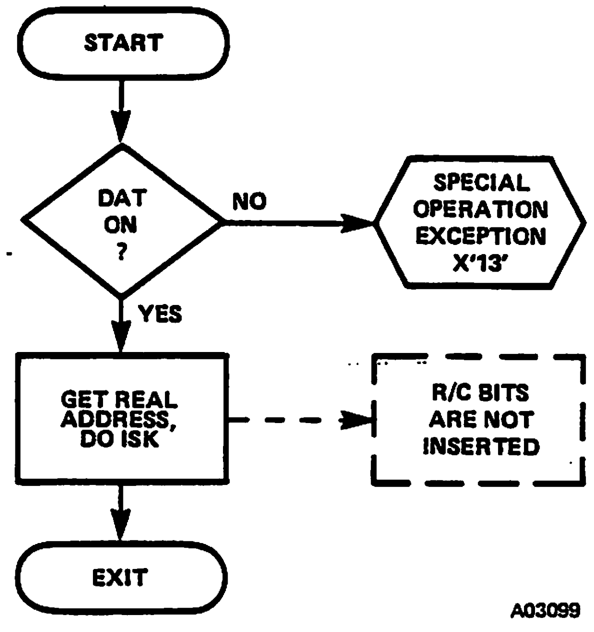

Figure 4-13. SPAB223A - Insert Virtual Storage Key Simulation

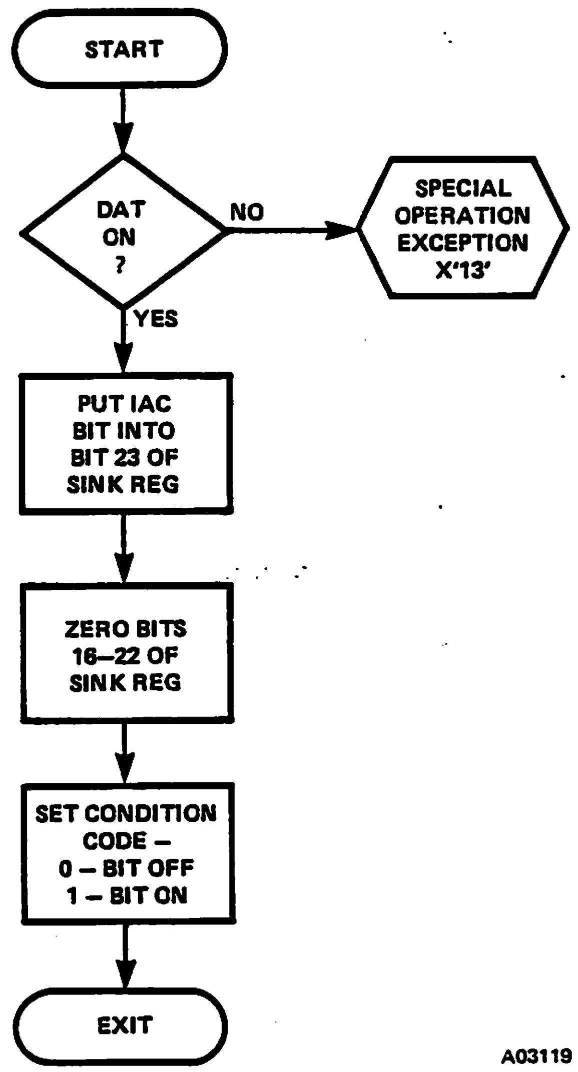
START
DAT ON ?
NO
SPECIAL OPERATION EXCEPTION X'13'
YES
PUT IAC BIT INTO BIT 23 OF SINK REG
ZERO BITS 16–22 OF SINK REG
SET CONDITION CODE – 0 – BIT OFF 1 – BIT ON
EXIT
A03119

Figure 4-14. SPAB224A - Insert Address Space Control Simulation

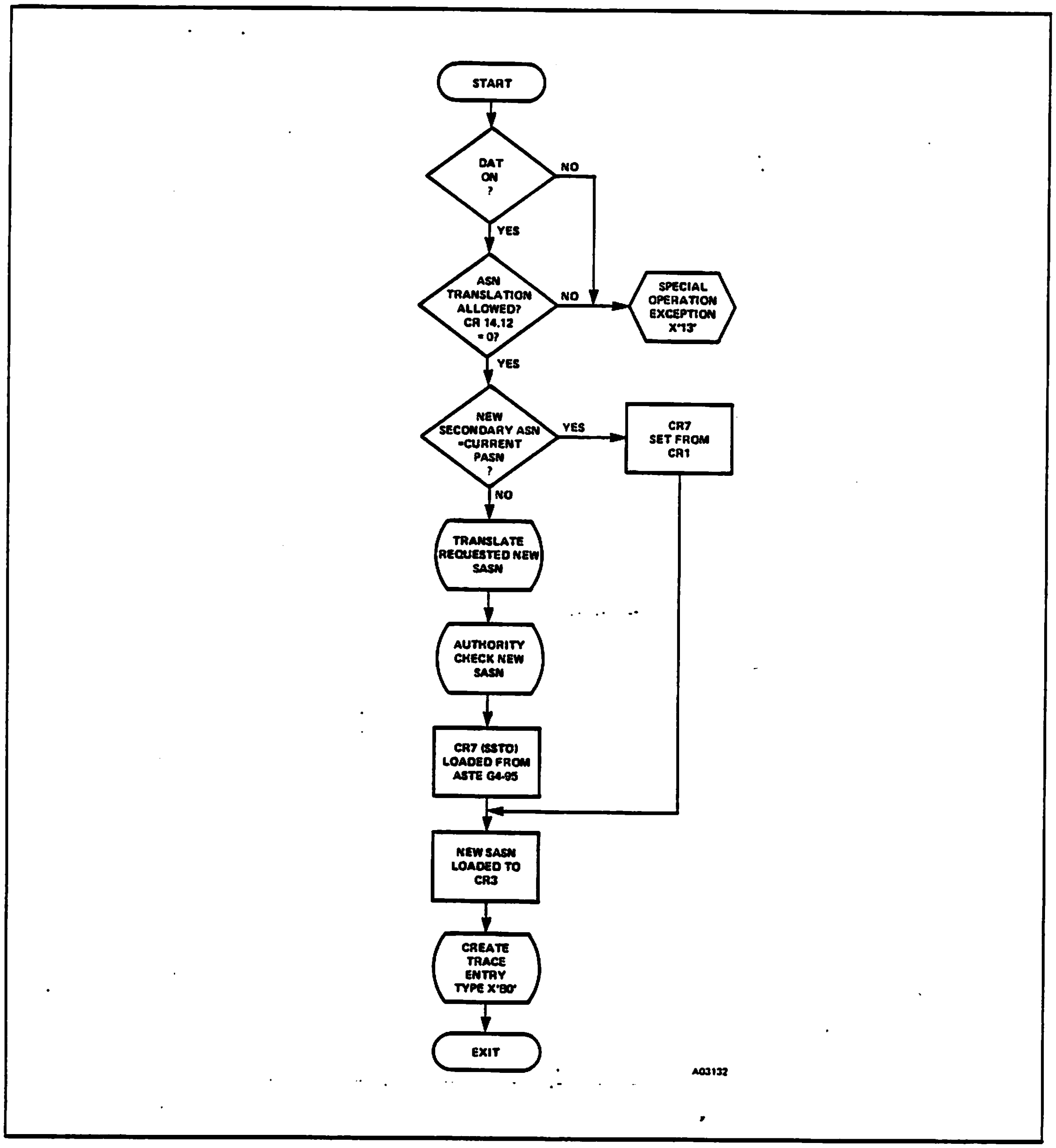
START
DAT ON ?
NO
YES
ASN TRANSLATION ALLOWED? CR 14.12 = 0?
NO
SPECIAL OPERATION EXCEPTION X'13'
YES
NEW SECONDARY ASN =CURRENT PASN ?
YES
CR7 SET FROM CR1
NO
TRANSLATE REQUESTED NEW SASN
AUTHORITY CHECK NEW SASN
CR7 (SSTO) LOADED FROM ASTE G4-95
NEW SASN LOADED TO CR3
CREATE TRACE ENTRY TYPE X'80'
EXIT
A03132

Figure 4-15. SPAB225A - Set Secondary ASN Simulation

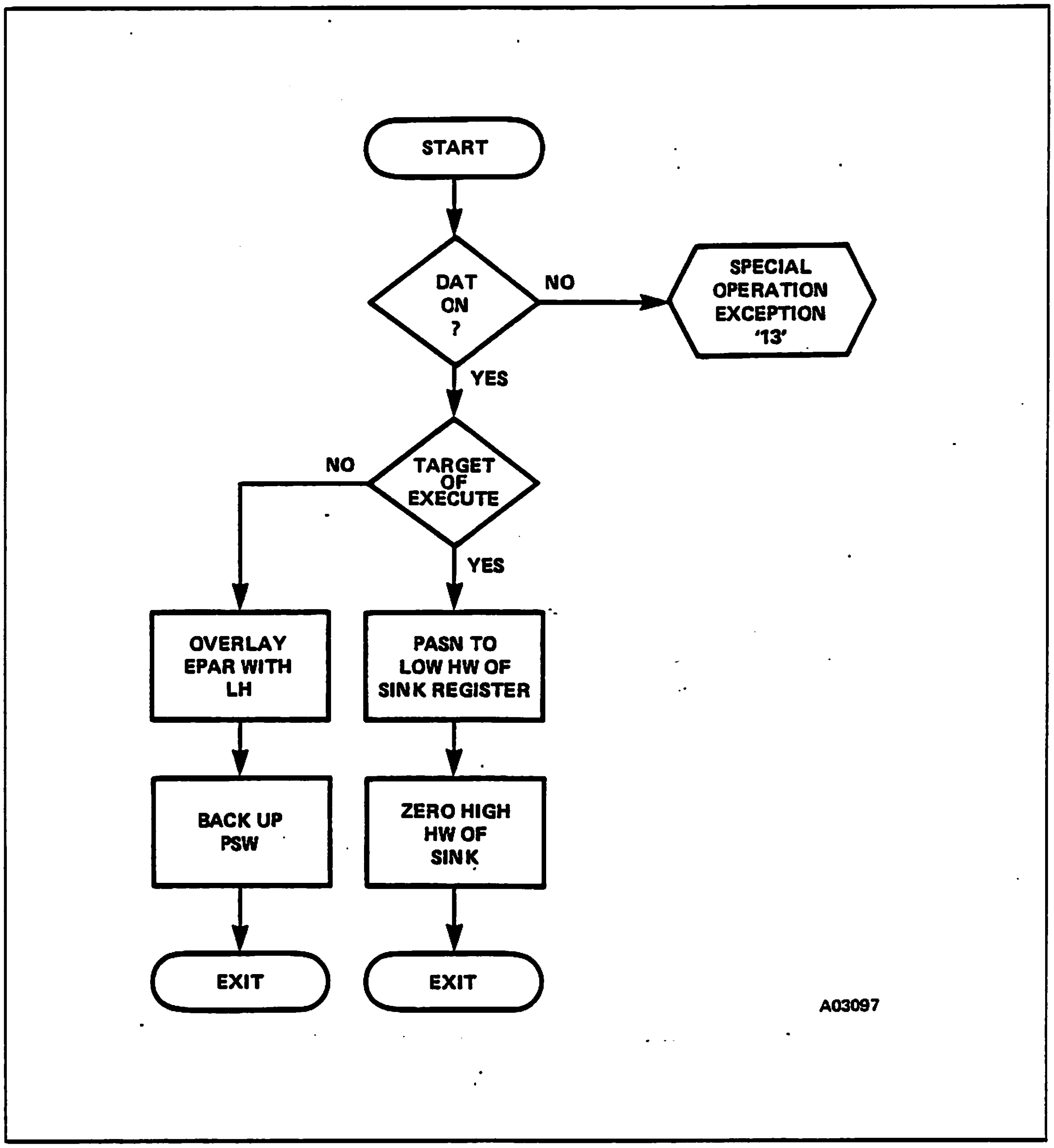

Figure 4-16. SPAB226A - Extract Primary ASN Processing

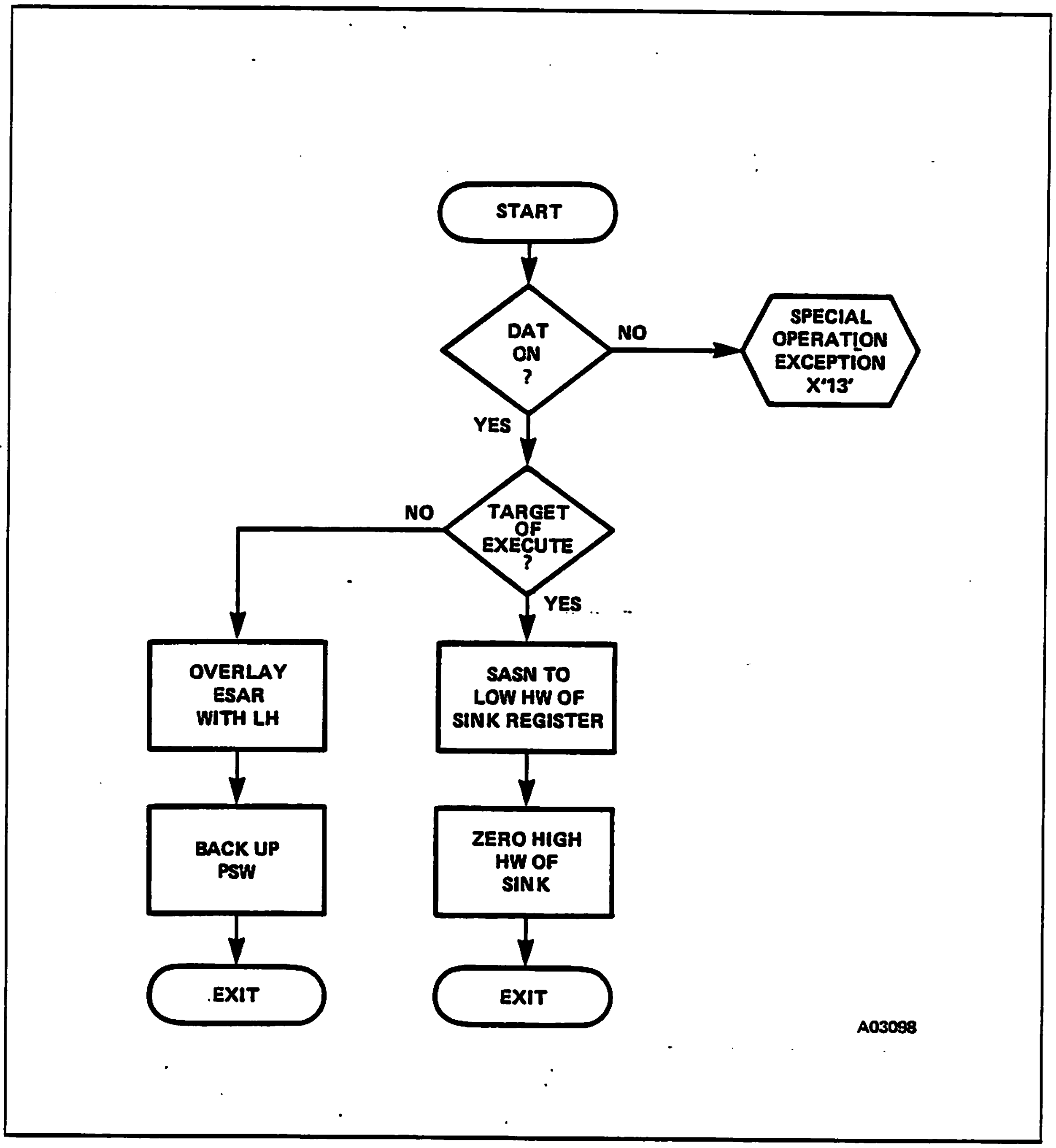

Figure 4-17. SPAB227A - Extract Secondary ASN Processing

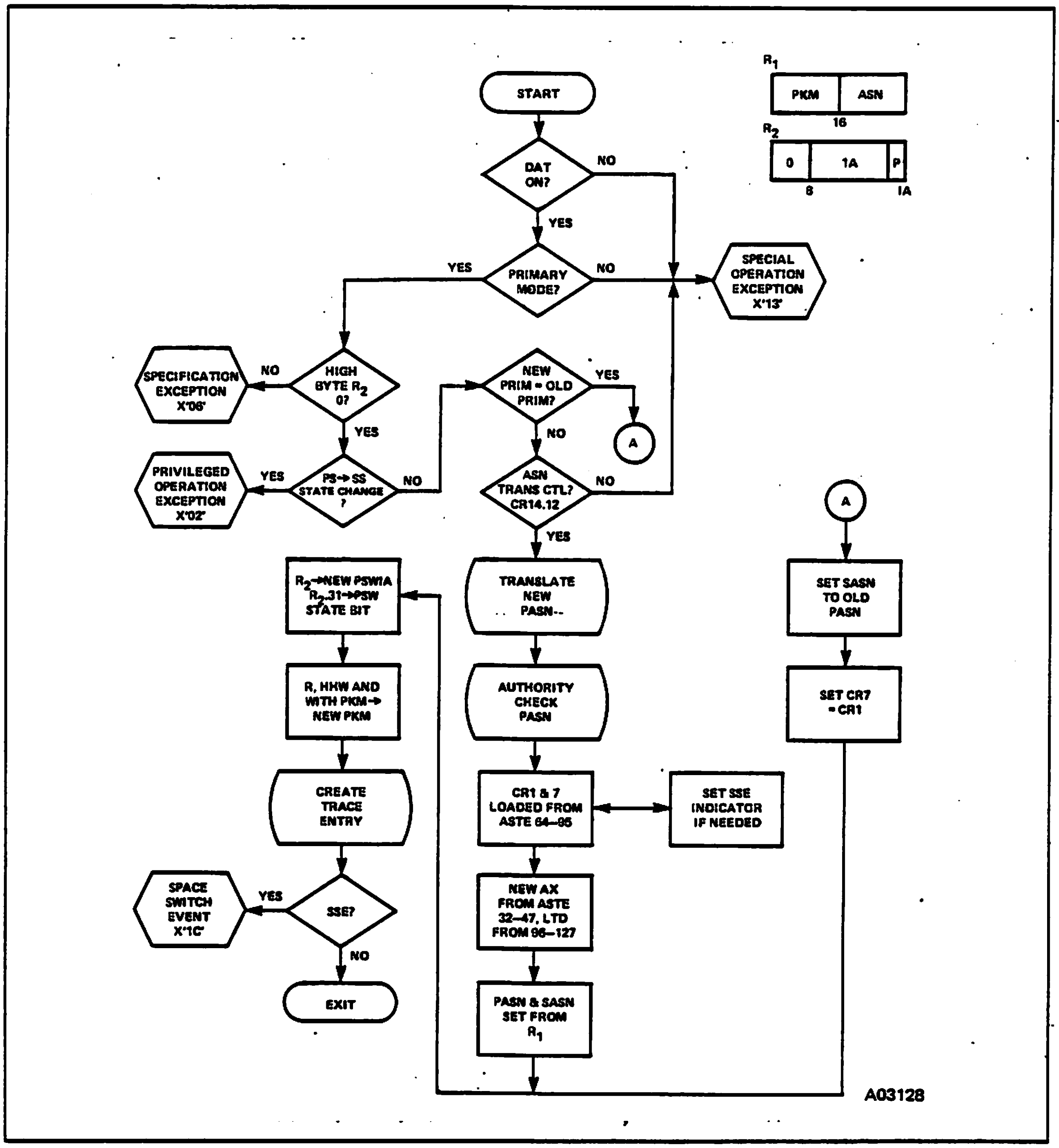

Figure 4–18. SPAB228A - Program Transfer Simulation

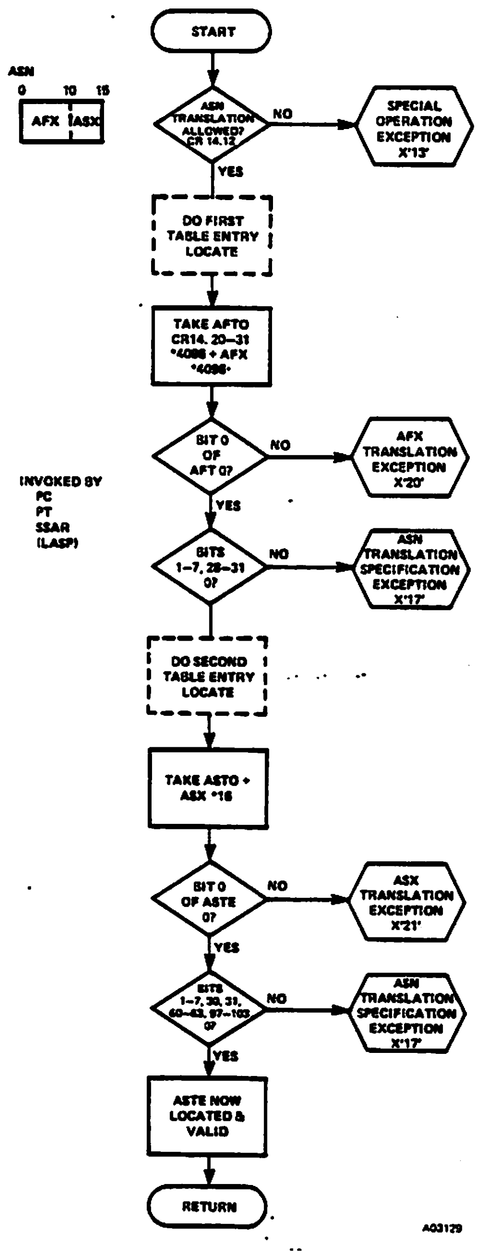
ASN
0 10 15
AFX
ASX
START
ASN TRANSLATION ALLOWED? CR 14.12
NO
YES
SPECIAL OPERATION EXCEPTION X'13'
DO FIRST TABLE ENTRY LOCATE
TAKE AFTO CR14. 20–31 *4096 + AFX *4096*
BIT 0 OF AFT 0?
NO
YES
AFX TRANSLATION EXCEPTION X'20'
INVOKED BY
PC
PT
SSAR
(LASP)
BITS 1–7, 28–31 0?
NO
ASN TRANSLATION SPECIFICATION EXCEPTION X'17'
DO SECOND TABLE ENTRY LOCATE
TAKE ASTO + ASX *16
BIT 0 OF ASTE 0?
NO
YES
ASX TRANSLATION EXCEPTION X'21'
BITS 1–7, 30, 31, 60–63, 97–103 0?
NO
YES
ASN TRANSLATION SPECIFICATION EXCEPTION X'17'
ASTE NOW LOCATED & VALID
RETURN
A03129

Figure 4-19. ASN Translation

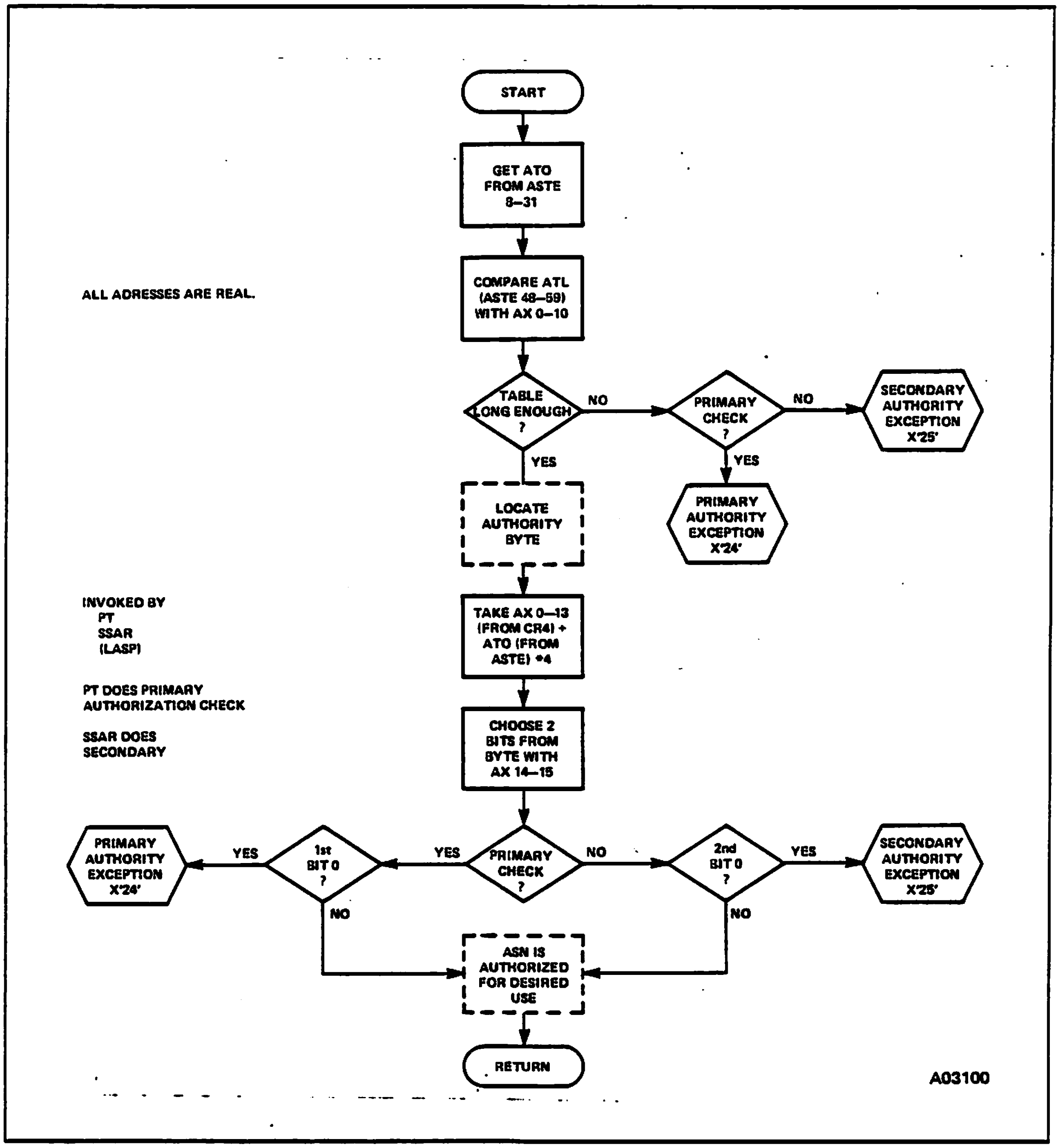
START
GET ATO FROM ASTE 8–31
COMPARE ATL (ASTE 48–59) WITH AX 0–10
ALL ADRESSES ARE REAL.
TABLE LONG ENOUGH ?
NO
YES
PRIMARY CHECK ?
NO
YES
SECONDARY AUTHORITY EXCEPTION X'25'
PRIMARY AUTHORITY EXCEPTION X'24'
LOCATE AUTHORITY BYTE
INVOKED BY PT SSAR (LASP)
PT DOES PRIMARY AUTHORIZATION CHECK
SSAR DOES SECONDARY
TAKE AX 0—13 (FROM CR4) + ATO (FROM ASTE) *4
CHOOSE 2 BITS FROM BYTE WITH AX 14–15
PRIMARY CHECK ?
YES
NO
1st BIT 0 ?
YES
NO
PRIMARY AUTHORITY EXCEPTION X'24'
2nd BIT 0 ?
YES
NO
SECONDARY AUTHORITY EXCEPTION X'25'
ASN IS AUTHORIZED FOR DESIRED USE
RETURN
A03100

Figure 4–20. ASN Authorization

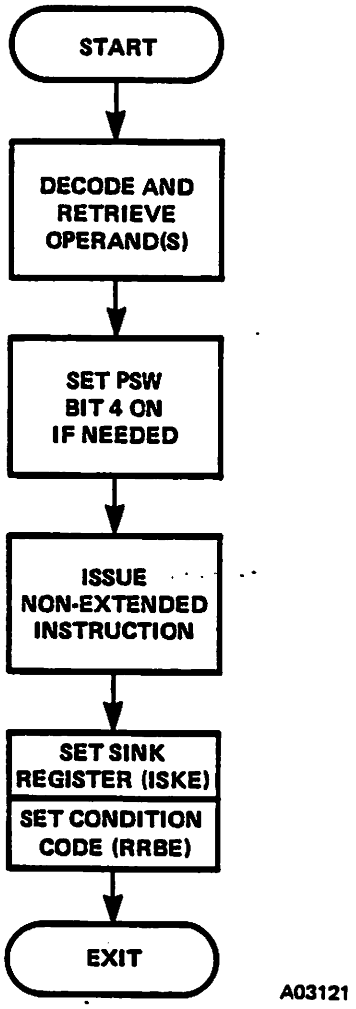
START
DECODE AND RETRIEVE OPERAND(S)
SET PSW BIT 4 ON IF NEEDED
ISSUE NON-EXTENDED INSTRUCTION
SET SINK REGISTER (ISKE)
SET CONDITION CODE (RRBE)
EXIT
A03121

Figure 4-21. SPAB229A-BA - Extended Storage Key Instruction Simulations

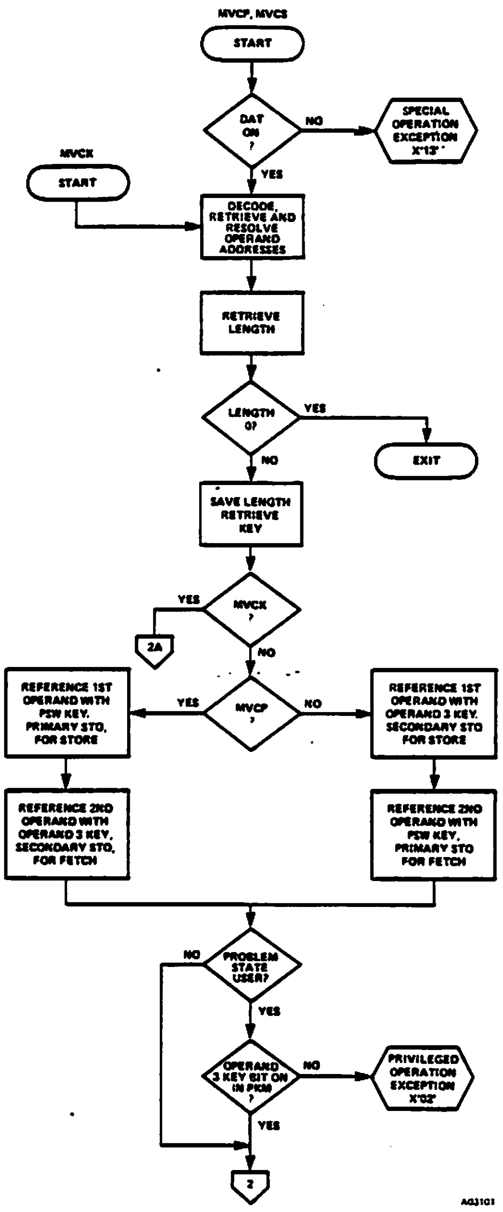
MVCP, MVCS
START
DAT ON ?
NO
SPECIAL OPERATION EXCEPTION X'13'
YES
MVCK
START
DECODE, RETRIEVE AND RESOLVE OPERAND ADDRESSES
RETRIEVE LENGTH
LENGTH 0?
YES
EXIT
NO
SAVE LENGTH RETRIEVE KEY
MVCK ?
YES
2A
NO
MVCP ?
YES
REFERENCE 1ST OPERAND WITH PSW KEY, PRIMARY STO, FOR STORE
NO
REFERENCE 1ST OPERAND WITH OPERAND 3 KEY, SECONDARY STO FOR STORE
REFERENCE 2ND OPERAND WITH OPERAND 3 KEY, SECONDARY STO, FOR FETCH
REFERENCE 2ND OPERAND WITH PSW KEY, PRIMARY STO FOR FETCH
PROBLEM STATE USER?
NO
YES
OPERAND 3 KEY BIT ON IN PKM ?
NO
PRIVILEGED OPERATION EXCEPTION X'02'
YES
2
A03101

Figure 4-22. MVCXCOMM - Move to Secondary/Move to Primary Simulation (1 of 3)

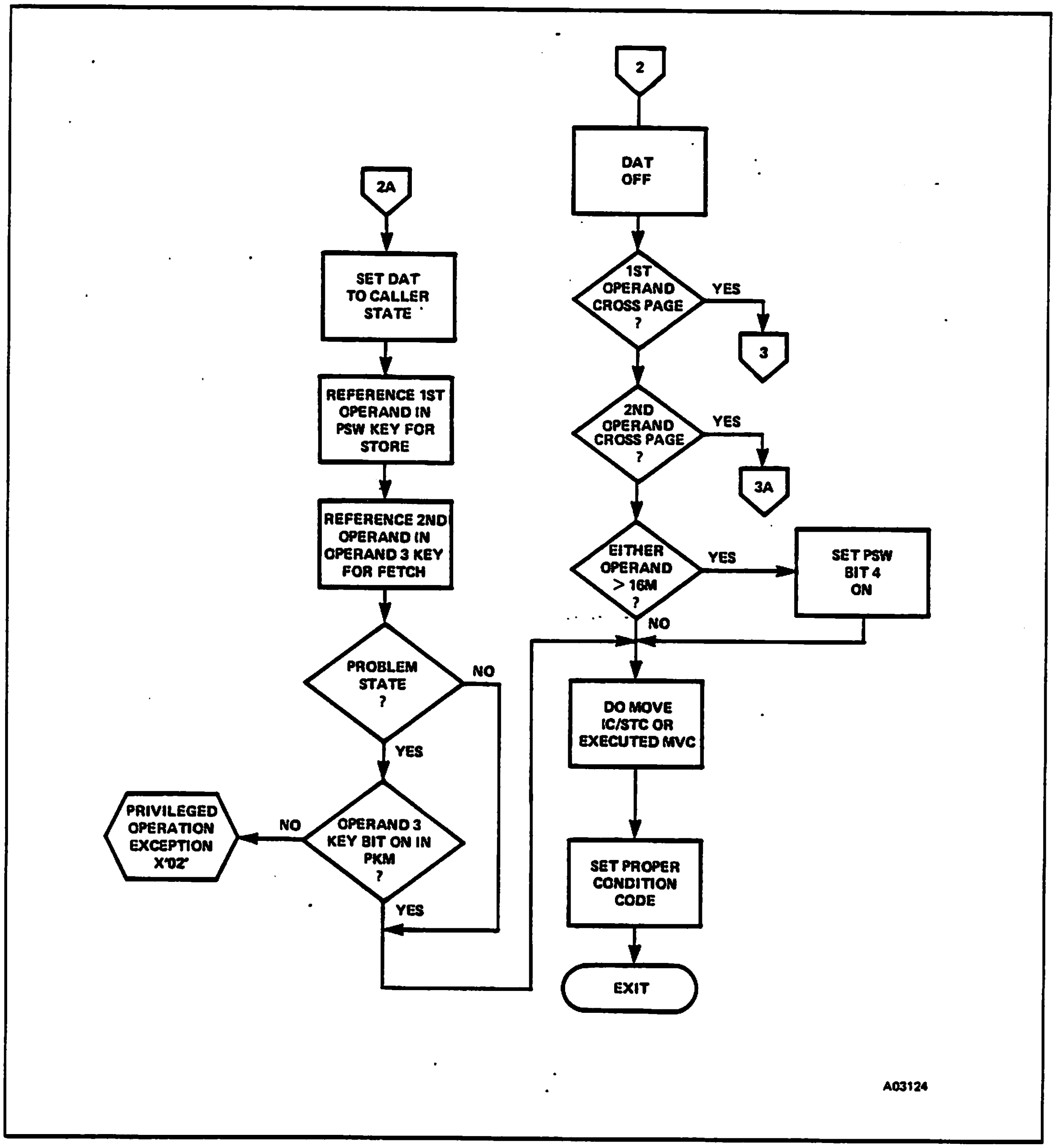
2
DAT OFF
1ST OPERAND CROSS PAGE ?
YES
3
2ND OPERAND CROSS PAGE ?
YES
3A
EITHER OPERAND > 16M ?
YES
SET PSW BIT 4 ON
NO
DO MOVE IC/STC OR EXECUTED MVC
SET PROPER CONDITION CODE
EXIT
2A
SET DAT TO CALLER STATE
REFERENCE 1ST OPERAND IN PSW KEY FOR STORE
REFERENCE 2ND OPERAND IN OPERAND 3 KEY FOR FETCH
PROBLEM STATE ?
NO
YES
OPERAND 3 KEY BIT ON IN PKM ?
NO
YES
PRIVILEGED OPERATION EXCEPTION X'02'
A03124

Figure 4-22. MVCXCOMM - Move to Secondary/Move to Primary Simulation (2 of 3)

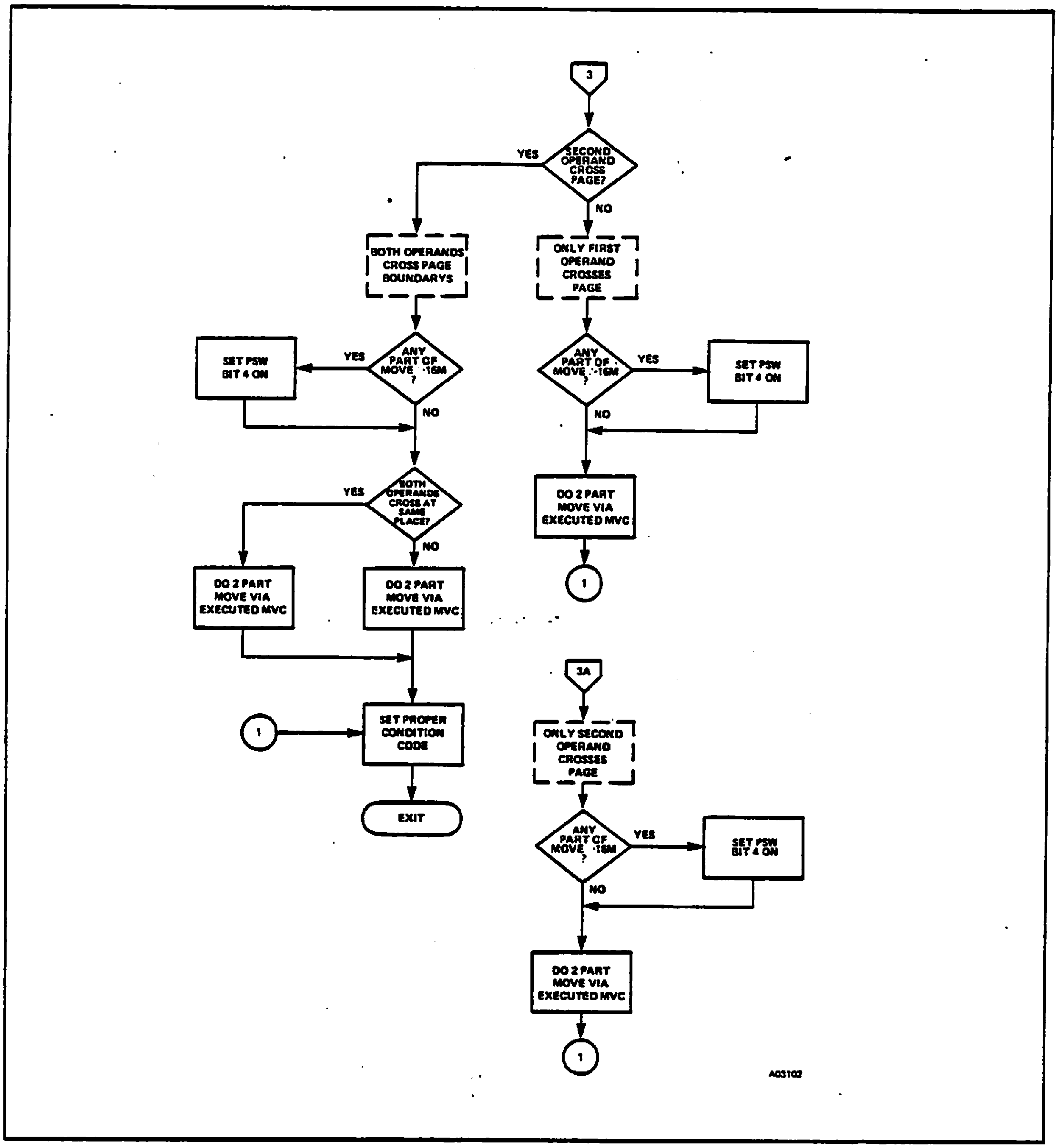
3
SECOND OPERAND CROSS PAGE?
YES
NO
BOTH OPERANDS CROSS PAGE BOUNDARYS
ONLY FIRST OPERAND CROSSES PAGE
ANY PART OF MOVE >16M ?
YES
NO
SET PSW BIT 4 ON
ANY PART OF MOVE >16M ?
YES
NO
SET PSW BIT 4 ON
BOTH OPERANDS CROSS AT SAME PLACE?
YES
NO
DO 2 PART MOVE VIA EXECUTED MVC
DO 2 PART MOVE VIA EXECUTED MVC
DO 2 PART MOVE VIA EXECUTED MVC
1
1
SET PROPER CONDITION CODE
EXIT
3A
ONLY SECOND OPERAND CROSSES PAGE
ANY PART OF MOVE >16M ?
YES
NO
SET PSW BIT 4 ON
DO 2 PART MOVE VIA EXECUTED MVC
1
A03102

Figure 4-22. MVCXCOMM - Move to Secondary/Move to Primary Simulation (3 of 3)

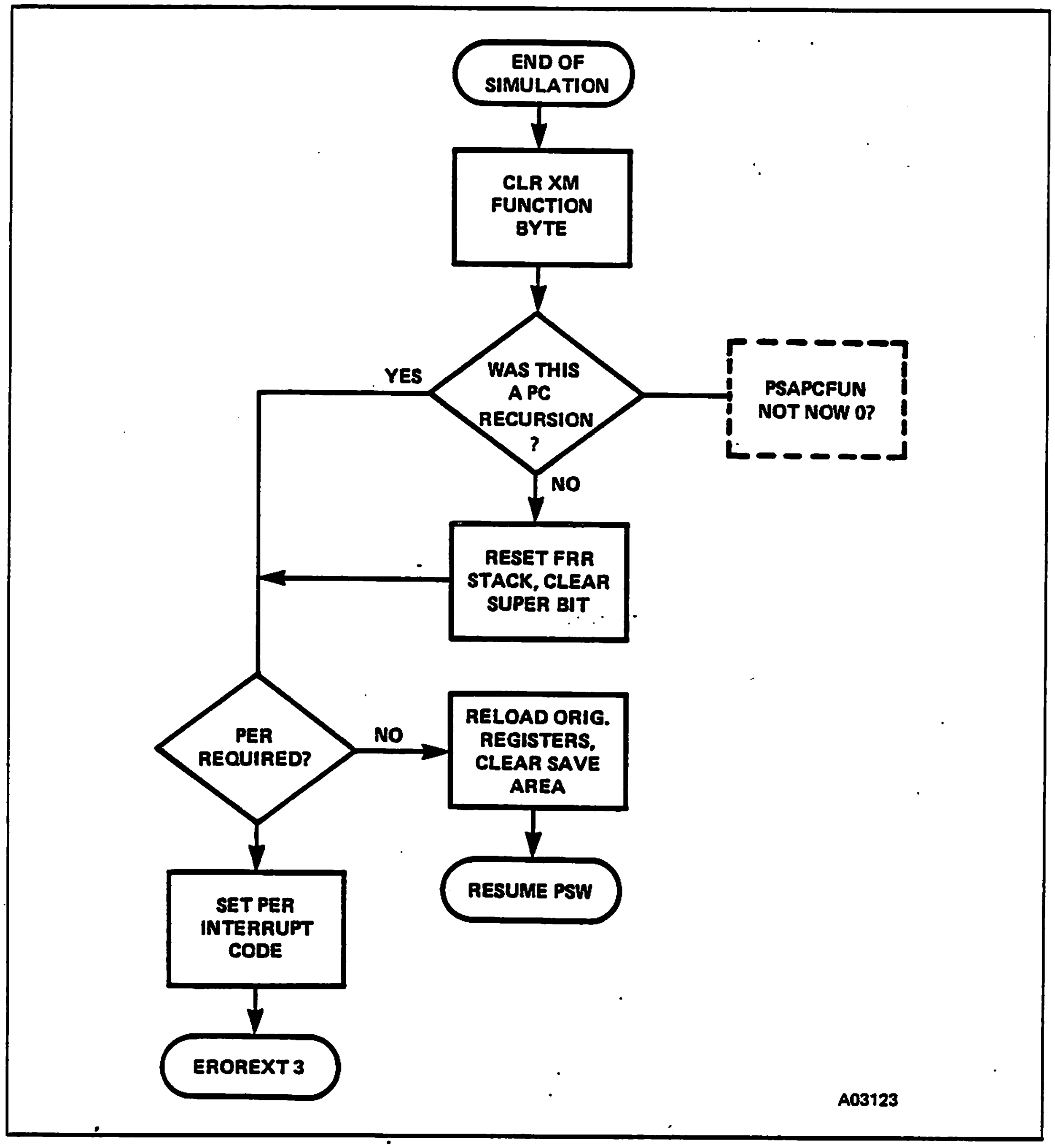

Figure 4-23. IEAVEXMS Resume Exit Processing

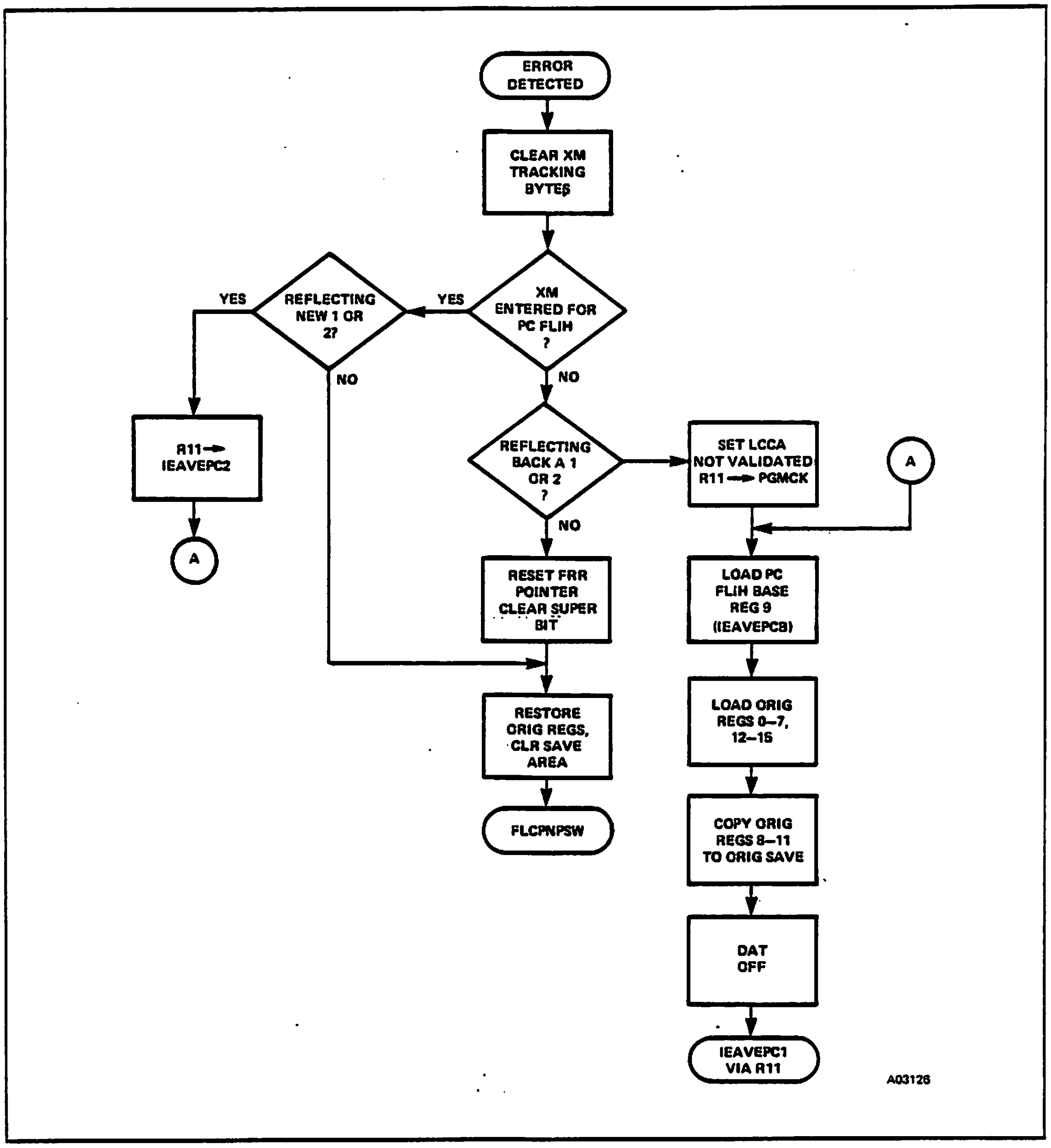
ERROR DETECTED
CLEAR XM TRACKING BYTES
XM ENTERED FOR PC FLIH ?
YES
NO
REFLECTING NEW 1 OR 2?
YES
NO
R11→ IEAVEPC2
A
REFLECTING BACK A 1 OR 2 ?
NO
SET LCCA NOT VALIDATED R11→ PGMCK
A
RESET FRR POINTER CLEAR SUPER BIT
RESTORE ORIG REGS, CLR SAVE AREA
FLCPNPSW
LOAD PC FLIH BASE REG 9 (IEAVEPCB)
LOAD ORIG REGS 0–7, 12–15
COPY ORIG REGS 8–11 TO ORIG SAVE
DAT OFF
IEAVEPC1 VIA R11
A03126

Figure 4-24. IEAVEXMS Error Exit

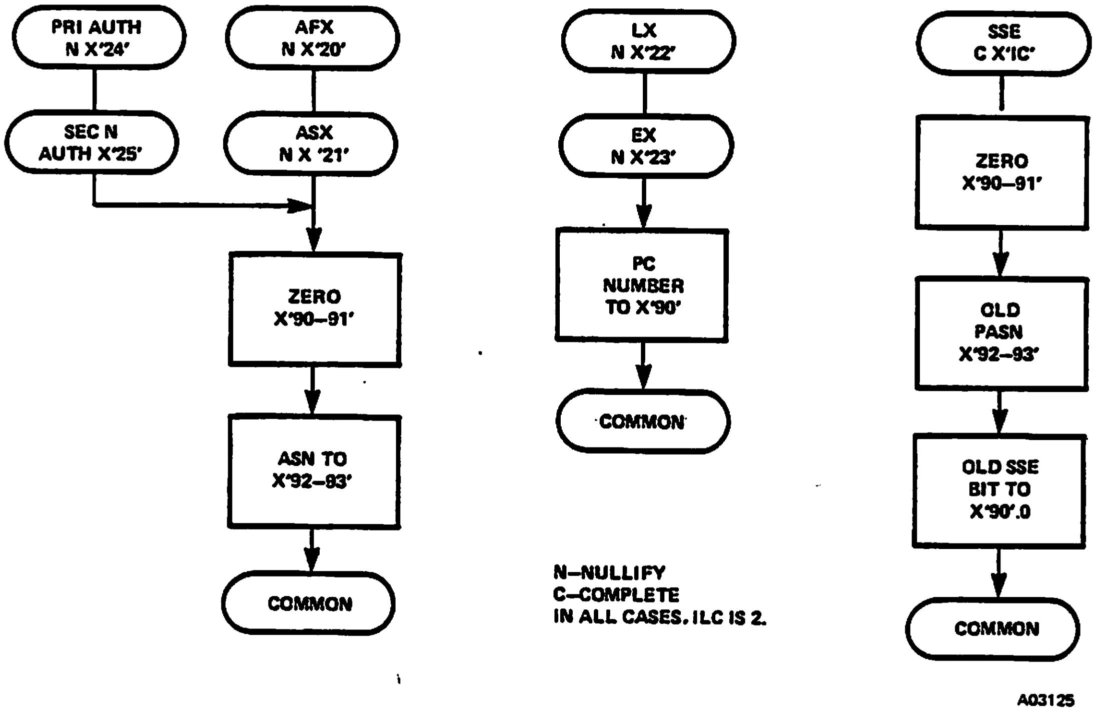

Figure 4-25. Special Program Exception Handling

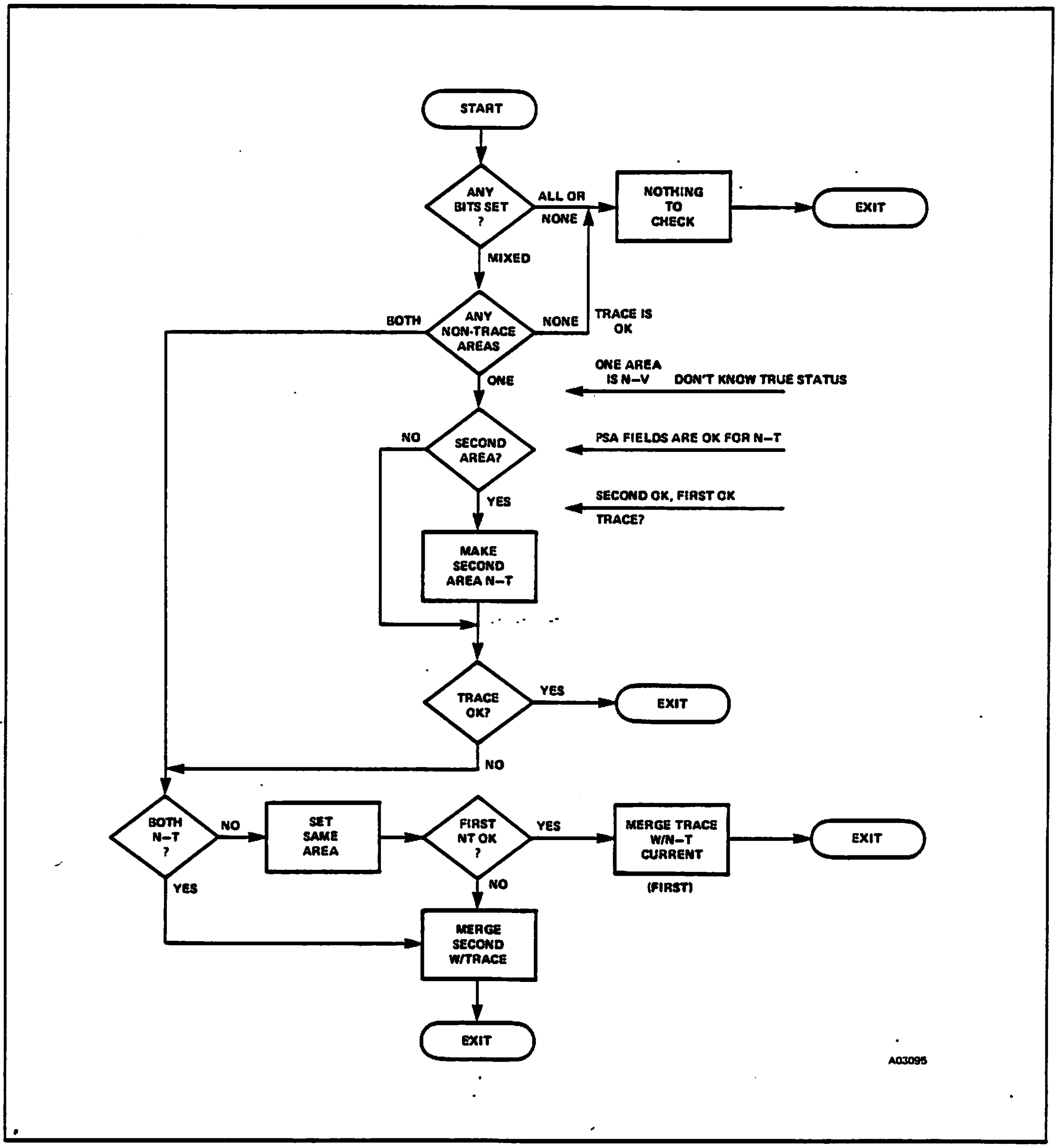

Figure 4-26. EF Instruction Non-Unique Trap Building

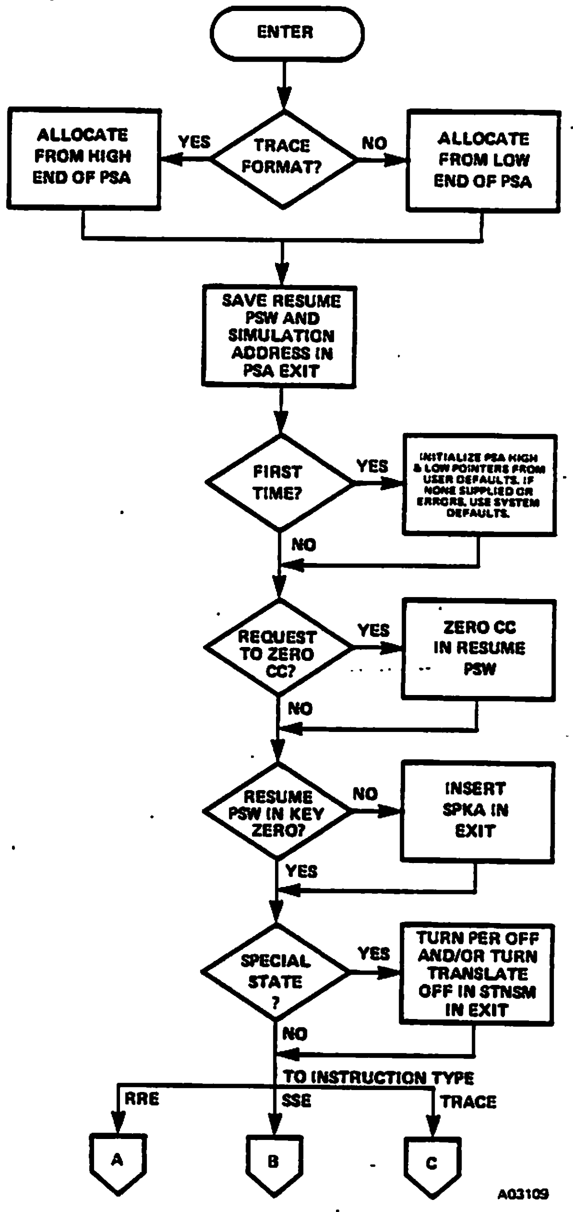
ENTER
TRACE FORMAT?
YES
ALLOCATE FROM HIGH END OF PSA
NO
ALLOCATE FROM LOW END OF PSA
SAVE RESUME PSW AND SIMULATION ADDRESS IN PSA EXIT
FIRST TIME?
YES
INITIALIZE PSA HIGH & LOW POINTERS FROM USER DEFAULTS. IF NONE SUPPLIED OR ERRORS, USE SYSTEM DEFAULTS.
NO
REQUEST TO ZERO CC?
YES
ZERO CC IN RESUME PSW
NO
RESUME PSW IN KEY ZERO?
NO
INSERT SPKA IN EXIT
YES
SPECIAL STATE ?
YES
TURN PER OFF AND/OR TURN TRANSLATE OFF IN STNSM IN EXIT
NO
TO INSTRUCTION TYPE
RRE
SSE
TRACE
A
B
C
A03109

Figure 4-27. Common Unique Trap Processing

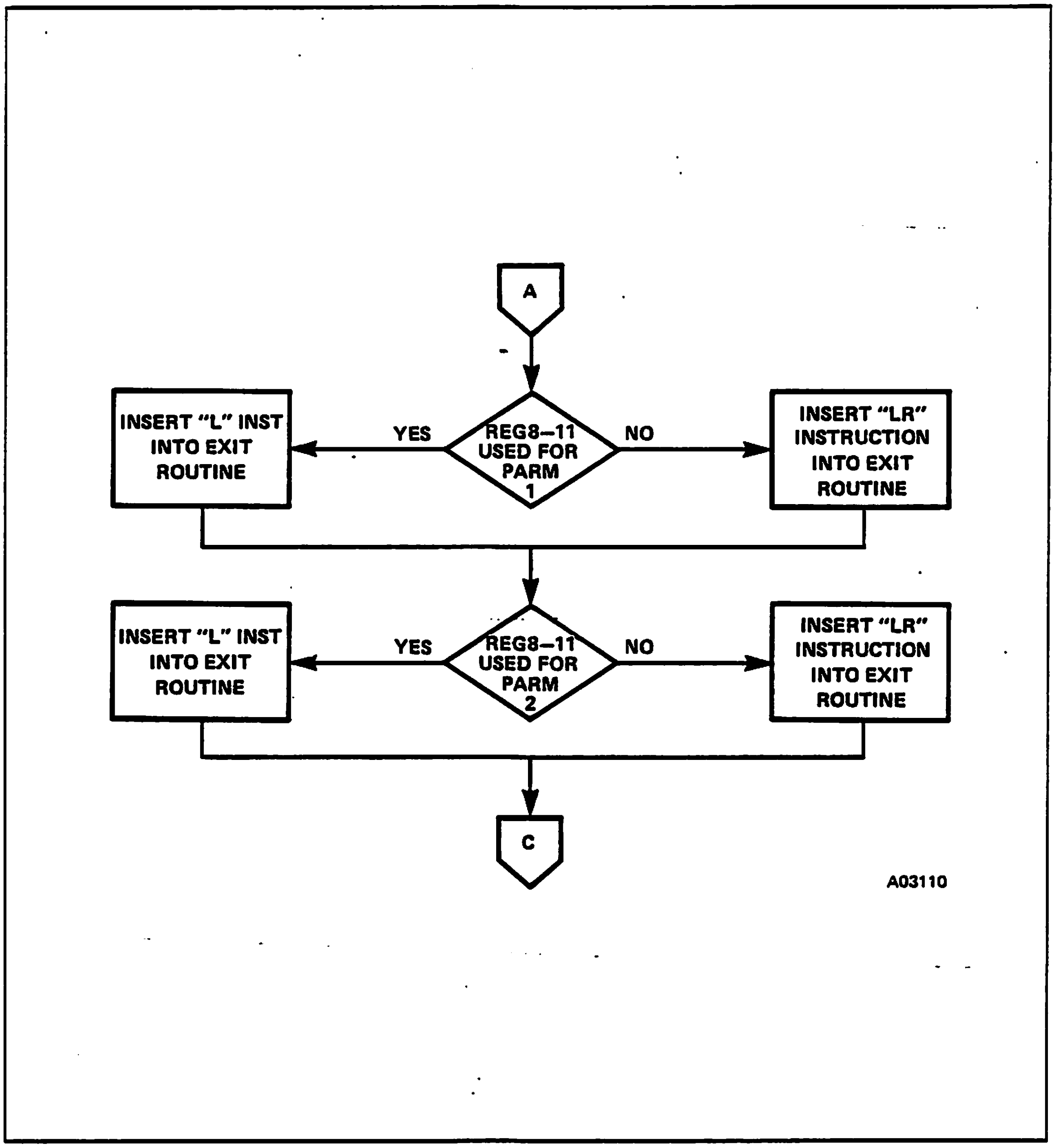
A
REG8–11 USED FOR PARM 1
YES
NO
INSERT "L" INST INTO EXIT ROUTINE
INSERT "LR" INSTRUCTION INTO EXIT ROUTINE
REG8–11 USED FOR PARM 2
YES
NO
INSERT "L" INST INTO EXIT ROUTINE
INSERT "LR" INSTRUCTION INTO EXIT ROUTINE
C
A03110

Figure 4-28. RRE Unique Trap Processing

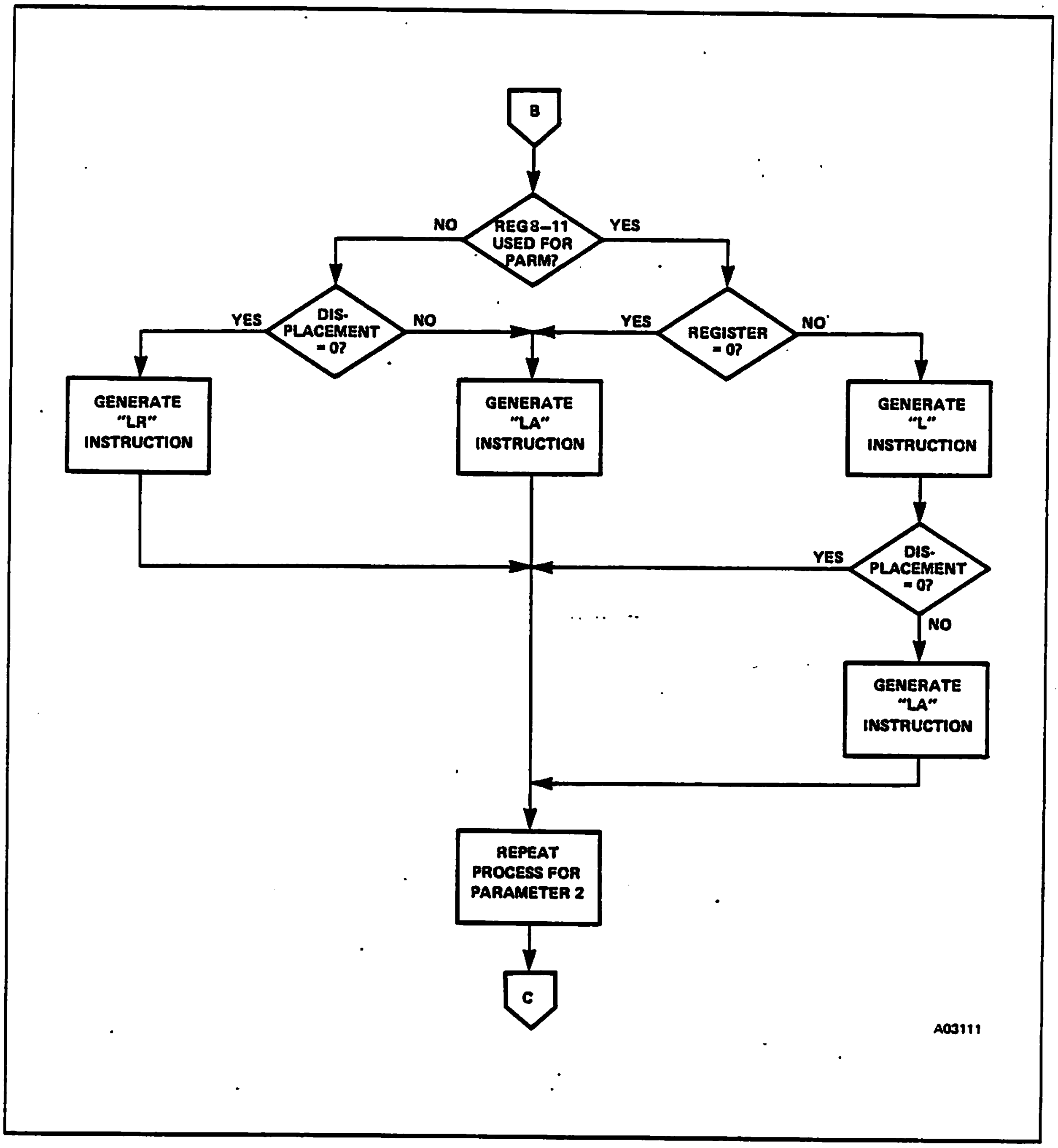

Figure 4–29. SSE Unique Trap Processing

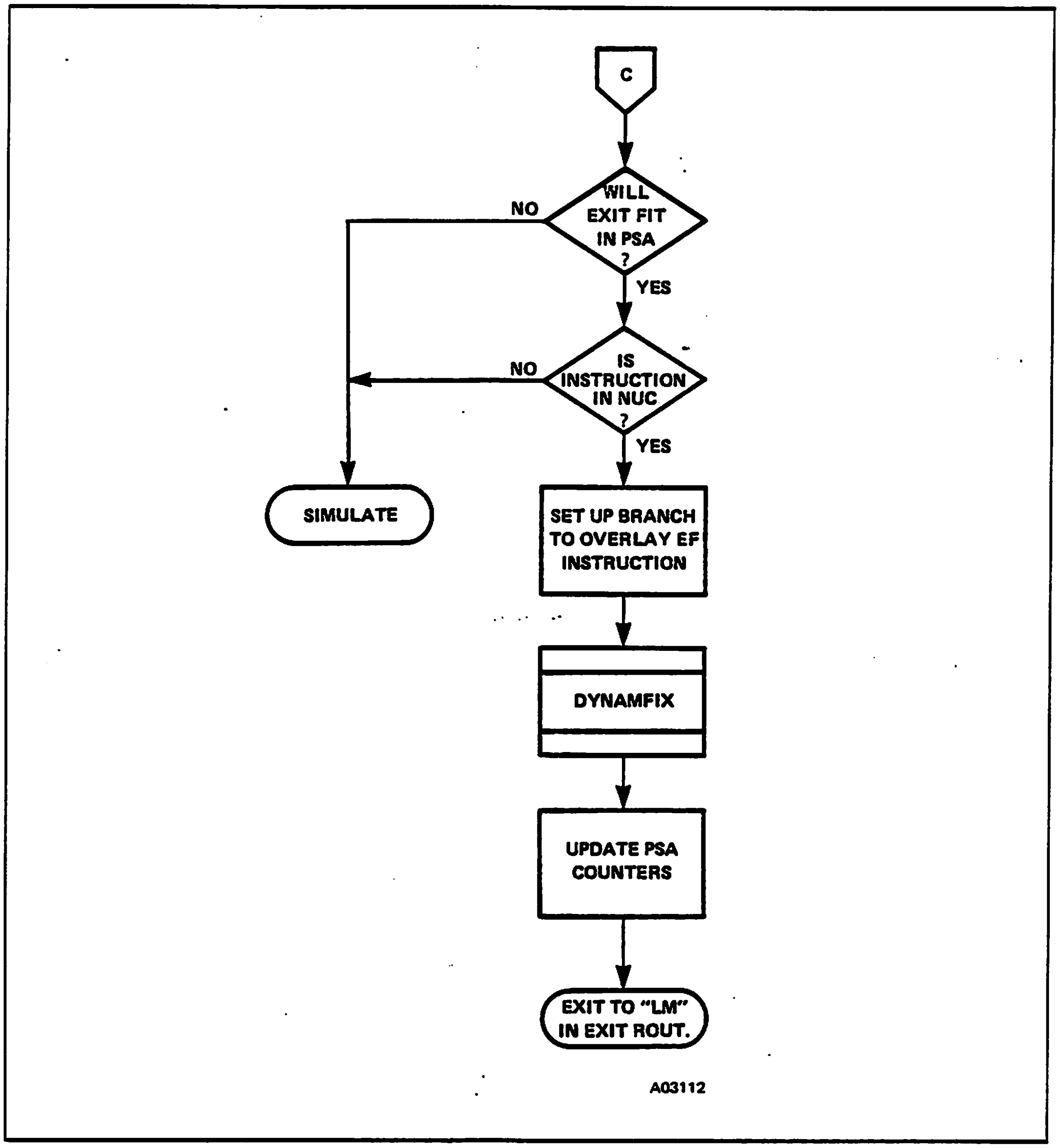

Figure 4-30. Trap Completion

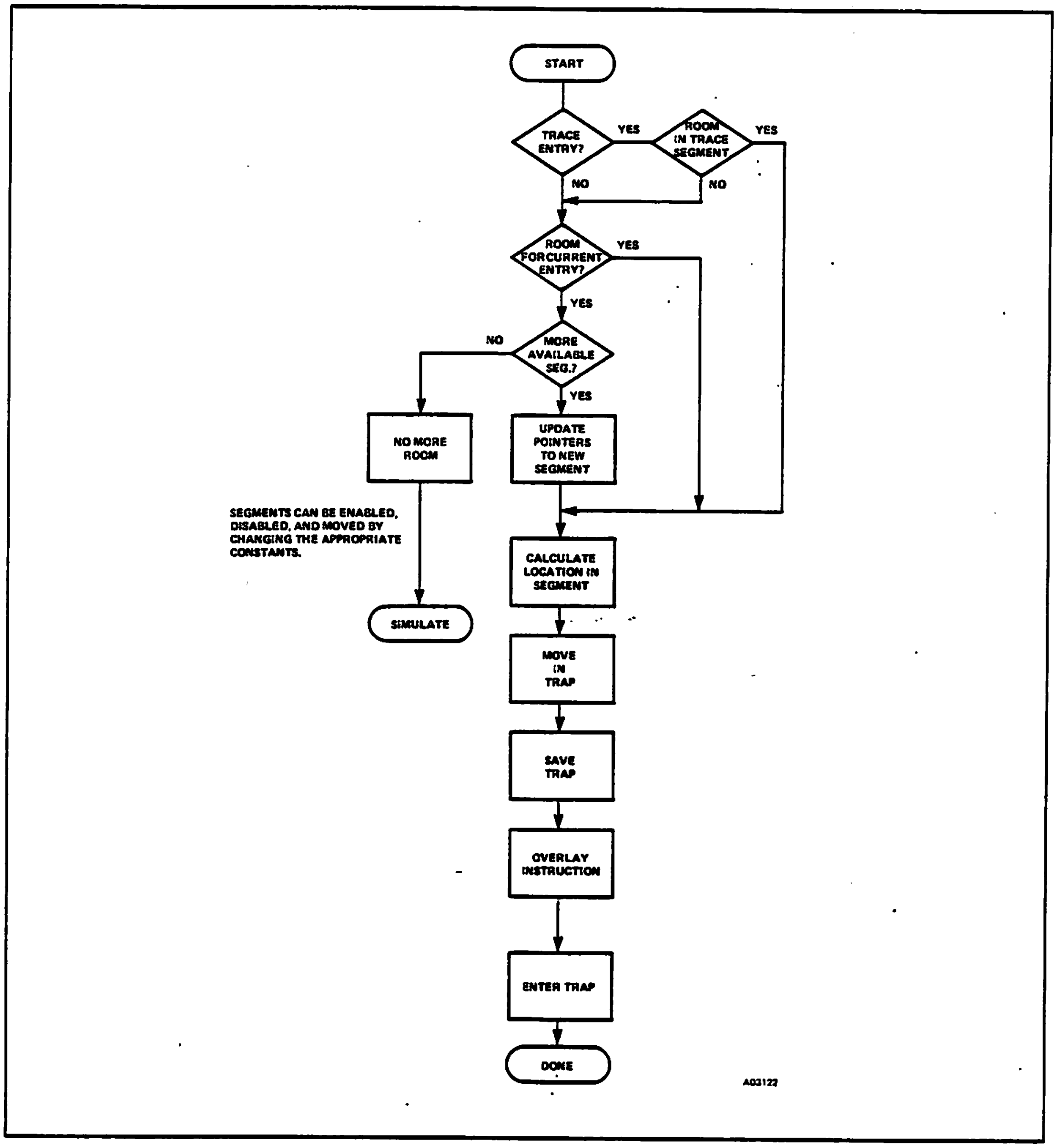

Figure 4-31. PSA Allocation

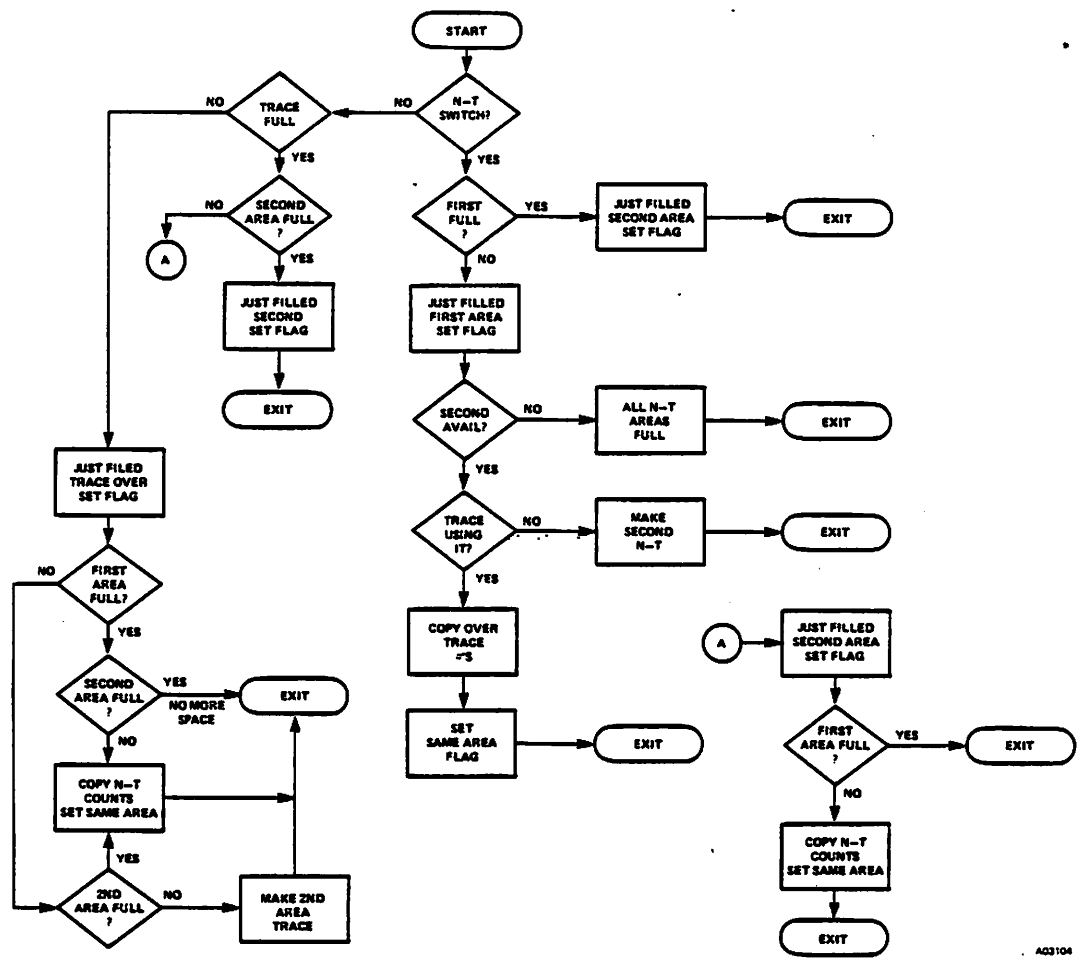

Figure 4-32. OVERRUN - Dynamic Patch Area Switching

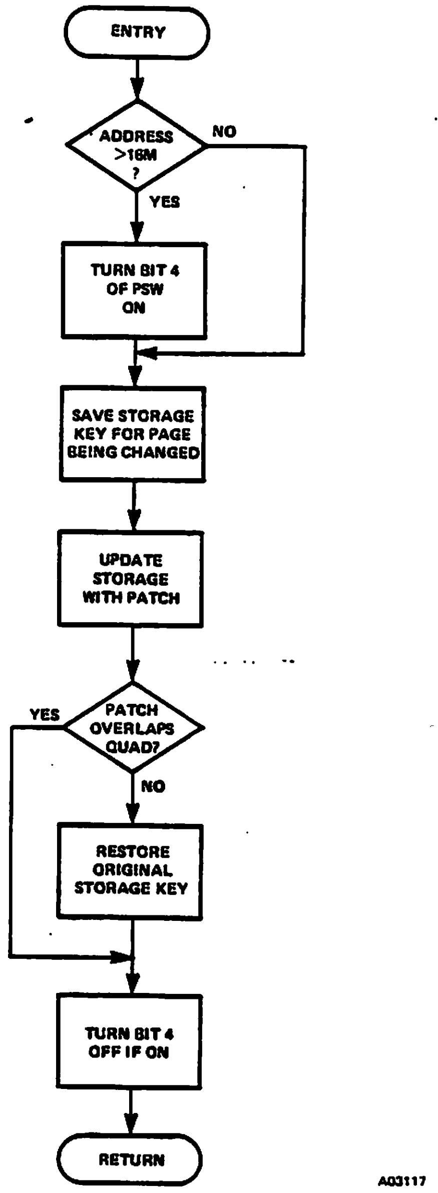

Figure 4-33. Dynamic Instruction Patching

APPENDIX A – DYNAMICALLY CONSTRUCTED CODE

Routine PSAALLOC replaces some instructions with linkage instructions to simulation routines. It also constructs exit routines in the PSA free area. The following paragraphs explain the logic of this dynamically constructed code.

A.1 DYNAMIC OVERLAYS

Some instructions are replaced by branches to the simulation routines, by branches to exit routines, by NO-OPs or by functionally equivalent instructions. Table A-1 shows the replacement code for each such instruction.

Table A–1. Instruction Replacement Code

OP CODE	REPLACEMENT	FUNCTION
B221	B PSA+n	To Unique Exit Routine
B226	LH Rx,PSAXMPAS	Set the Primary ASN
B227	LH Rx,PSAXMSAS	Set the Secondary ASN
E501	B PSA+n	To Unique Exit Routine
E502	BAL R14,PSAPFXA DC S(D1(B1))	To Page Fix Assist Prologue
E503	LR R0,R0 LR R1,R1 LR R2,R2	NOPs
E504:E507	L R13,PSALEXIT+n BALR R12,R13	To Simulation Routine
E508:E50D	B PSA+n	To Unique Exit Routine

A.2 PAGE FIX ASSIST PROLOGUE

The PSA has an area reserved for the following instruction sequence:

```
PSAPFXA   STM  R8,R11,PSAXMGR8        Save Registers
          L    R11,PSAPFXAA           Get Routine Address
          B    0(,R11)                Call It
PSAPFXAA  DC   A(SPAE502A)            Routine Address
```

The first time a FIX PAGE instruction is encountered, the instructions above are moved into the PSA at label PSAPFXA. This code is the 'Page Fix Prologue'. The E502 is then replaced, as shown in table A-1, by a branch and link to the prologue, followed by the first operand of the replaced instruction. The prologue saves the work registers in the XMS work area, loads the Page Fix Assist routine's base address, and branches to the routine.

The Fix Page prologue is used for all FIX PAGE instructions.

A.3 SVC ASSIST NOPS

Table A-1 shows that the SVC ASSIST instruction (E503) is replaced by LR NOPs. This technique is used because MVS does not require the SVC Assist instructions be issued or that any SVC be assisted. The SVC call causing the SVC Assist instruction is treated as 'non-assistable'. The referenced IBM System/370 Assists for MVS manual explains the difference between assistable and non-assistable SVC calls.

The LR Rx,Rx instructions are used because they do not alter any registers, occupy two bytes each, do not change the condition code, require minimum execution time, and do not disrupt the pipeline processing of the following instruction stream.

A.4 LINKAGE TO LOCK-HANDLING INSTRUCTIONS

Table A-1 shows the linkage code for for the lock-handling instructions (E504-E507). The LOAD instruction loads register 13 with the address of the appropriate lock-handling instruction simulation routine. The BALR instruction puts the return address into register 12 and calls the simulation routine.

A.5 EXIT ROUTINES

The Test Protect, IPTE and trace instructions cause the building of exit routines in the PSA free area. These exit routines are similar to the Fix Page prologue; however, they are more complex and can be used only for a specific occurrence of an EF instruction routine.

The exact contents of an exit routine depend on the operand addresses of the associated EF instruction and on the PSW at the time the EF instruction is issued. All exit routines have the following general format:

```
   0    program check ...             Saved as Resume
   4    ... old PSW                   PSW
   8    A(simulation routine)         Routine to be Called
   C    STNSM    byte1,x'BC'          Disable Interrupts and PER
  10    STM      R8,R11,PSAXMGR8      Save Registers 8-11
  14    LM       R8,R10,oldPSW        Get Resume PSW and Routine Base
  18    STM      R8,R9,PSAXMPSW       Set Resume PSW
  1C    BR       R10                  Call Simulation Routine
```

The 'BR R10' is replaced with 'BAL R11,0(,R10)' for an IPTE trap, to provide for restoring the condition code (from register 11) on exit from the simulation.

The eight word exit routine above is the simplest type of exit. It is used for the six trace instructions whose parameters can be located by the simulation routine.

Exit routines grow larger when the resume PSW in the exit has a non-zero protection key. In this case, a SPKA instruction is placed just before the STNSM instruction in the exit routine.

For the IPTE and Test Protect instructions, additional instructions are inserted before the final BR R10 to load the EF instruction's first and second operands into registers 8 and 9, respectively. The following instruction sequences are possible for each operand with base register b and displacement ddd:

```
LR    Rx,b           when 0<=b<=7 or 12<=b<=15 and ddd=0

L     Rx,PSAXMGRb    when 8<=b<=11 and ddd=0

L     Rx,PSAXMGRb    when 8<=b<=11 and ddd¬=0
LA    Rx,ddd(,Rx)

LA    Rx,ddd(,b)     when 0<=b<=7 or 12<=b<=15 and ddd¬=0
```

APPENDIX B — EXCEPTIONS GENERATED

Table B-1 provides a cross reference showing which simulations can generate which interrupts and what specific conditions cause the interrupt. A cross reference of interrupt code to abend code is also provided.

Other interrupts can also be generated, for example from bad input data or addresses. The occurrences listed here are explicitly generated by MVS/SPA.

Table B-1. Interrupts Caused by Simulations

SIMULATION	EXCEPTION	CAUSE
Non-Routine Oriented Exception	Operation	Opcode passed to IEAVEXMS is unsupported or invalid.
	Privileged Operation	An EF instruction was issued in problem state.
SPKA	Privileged Operation	The key the problem state caller attempted to set was not valid in the PSW key mask.
Program Call	Privileged Operation	The problem state caller's key was not valid in the ETE authorization key mask.
	Special Operation	DAT was not on. The caller was insecondary mode. Program call number translation was not allowed (CR5.0 was 0). ASN translation was not allowed (CR14.12 was 0).
	ASN Translation Specification	Bits 1-7 & 28-31 of the AFTE were not all zero. Bits 1-7, 30-31, 60-63 & 97-103 of the ASTE were not all zero.

(continued)

Table B-1. Interrupts Caused by Simulations (continued)

SIMULATION	EXCEPTION	CAUSE
	Space Switch Event	Bit 31 was on in either the old or new Primary STD.
	Program Call Translation Specification	Bits 1-7 of the LTE were not all zero. Bits 32-39 of the ETE were non-zero (first byte of ETE entry point address).
	AFX Translation	The ASTE invalid bit (bit 0) was on.
	ASX Translation	The ASTE invalid bit (bit 0) was on.
	Linkage Translation	The Program Call linkage table index (LTX) exceeded the linkage table length (LTL), bits 25-31 of CR5. The invalid bit (bit 0) was on in the linkage table entry (LTE).
	Entry Translation	The Program Call entry table index (ETX) exceeded the entry table length (ETL), bits 26-31 of the linkage table entry.
	Entry Translation	The ETX exceeded the ETL, bits 26-31 of the linkage table entry.
Set Address Space Control	Special Operation	DAT was not on.
	Specification	Bits 20-22 of the operand are not all zero.
Insert Virtual Storage Key	Special Operation	DAT was not on.
Insert Address Space Control	Special Operation	DAT was not on.

(continued)

Table B-1. Interrupts Caused by Simulations (continued)

SIMULATION	EXCEPTION	CAUSE
Set Secondary ASN	Special Operation	DAT was not on. ASN translation was not allowed (CR14.12 was 0).
	ASN Translation Specification	Bits 1-7 & 28-31 of the AFTE were not all zero. Bits 1-7, 30-31, 60-63 & 97-103 of the ASTE were not all zero.
	AFX Translation	The AFTE invalid bit (bit 0) was on.
	ASX Translation	The ASTE invalid bit (bit 0) was on.
	Secondary Authority	The authorization index (AX) exceeded the authorization table length (ASTE bits 48-59). The secondary bit for the caller's AX was zero.
Program Transfer	Privileged Operation	A problem state caller tried to transfer into supervisor state.
	Specification	High byte of the second operand register was not zero.
	Special Operation	DAT was not on. The caller was in secondary mode. ASN translation was not allowed (CR14.12 was 0).
	ASN Translation Specification	Bits 1-7 & 28-31 of the AFTE were not all zero. Bits 1-7, 30-31, 60-63 & 97-103 of the ASTE were not all zero.

(continued)

Table B-1. Interrupts Caused by Simulations (continued)

SIMULATION	EXCEPTION	CAUSE
	Space Switch Event	Bit 31 was on in either the old or new Primary STD.
	AFX Translation	The AFTE invalid bit (bit 0) was on.
	ASX Translation	The ASTE invalid bit (bit 0) was on.
	Primary Authority	The authorization index (AX) exceeded the authorization table length (ASTE bits 48-59). The primary bit for the caller's AX was zero.
Move to Primary	Privileged Operation	The key specified by the problem state caller in the operand 3 register was not valid in the caller's PSW key mask.
	Special Operation	DAT was not on.
Move to Secondary	Privileged Operation	The key specified by the problem state caller in the operand 3 register was not valid in the caller's PSW key mask.
	Special Operation	Dat was off.
Move with Key	Privileged Operation	The key specified by the problem state caller in the operand 3 register was not valid in the caller's PSW key mask.

Table B-2 contains the ABEND codes that would result from the specific program interrupt were it to pass through to RTM. Some of the interrupts are occasionally expected; not every program check with one of these codes results in the ABEND.

The X'26' interrupt code is included only for completeness. It should never occur on a system without the 3033 Extension microcode installed.

Table B–2. Cross Reference of Interrupt Code to ABEND Code

INTERRUPT CODE (HEX)	INTERRUPT	ABEND
02	Privileged Operation Exception	0C2
04	Protection Exception	0C4
06	Specification Exception	0C6
13	Special Operation Exception	0D3
17	ASN Translation Exception	0D4
1C	Space Switch Event	0D8
1F	Program Call Translation Exception	0DA
20	AFTE Translation Exception	0D5
21	ASTE Translation Exception	0D5
22	LX Translation Exception	0D6
23	EX Translation Exception	0D6
24	Primary Authorization Exception	0D7
25	Secondary Authorization Exception	0D7
26	Page Fault Assist Failed	0D9

APPENDIX C - INSTRUCTION TRAP/SIMULATION CROSS REFERENCE

Table C-1 provides a cross reference of which routine simulates a given instruction and which routine, if any, constructs the specific trap for the instruction. All trap routines are processed by PSAALLOC, so the entry in the 'TRAP BUILD' column is for the routine that sets the operation parameters for PSAALLOC.

In general the routine names are SPAxxxxy, where xxxx is the opcode and y is blank if the routine is called directly from a PSA trap or instruction patch, and 'A' where it is called from the routing tables in IEAVEXMS.

Table C-1. Cross Reference of Instruction Trap/Simulation

OPCODE	INSTRUCTION	SIMULATION	TRAP BUILD
B218	Program Call	SPAB218A	
B219	Set Address Space Control	SPAB219A	
B221	Invalidate Page Table Entry	SPAB218A	
B223	Insert Virtual Storage Key	SPAB223A	
B224	Insert Address Space Control	SPAB224A	
B225	Set Secondary ASN	SPAB225A	
B226	Extract Primary ASN		SPAB226A
B227	Extract Secondary ASN		SPAB227A
B228	Program Transfer	SPAB228A	
B229	Insert Storage Key Extended	SPAB229A	
B22A	Reset Reference Bit Extended	SPAB22AA	
B22B	Set Storage Key Extended	SPAB22BA	
D9	Move with Key	MVCKCOMM	
DA	Move to Primary	MVCXCOMM	
DB	Move to Secondary	MVCXCOMM	
E501	Test Protection	SPAE501	SPAE501A
E502	Page Fix Assist	SPAE502	SPAE502A
E503	SVC Assist		SPAE503A
E504	Obtain Local Lock	SPAE504	SPAE504A
E505	Release Local Lock	SPAE505	SPAE505A
E506	Obtain CMS Lock	SPAE506	SPAE506A
E507	Release CMS Lock	SPAE507	SPAE507A
E508	Trace SVC Interrupt	SPAE508	FIXINSTR
E509	Trace Program Interrupt	SPAE509	FIXINSTR
E50A	Trace Initial SRB Dispatch	SPAE50A	FIXINSTR
E50B	Trace I/O Interrupt	SPAE50B	FIXINSTR
E50C	Trace Task Dispatch	SPAE50C	FIXINSTR
E50D	Trace SVC Return	SPAE50D	FIXINSTR

APPENDIX D - MVS/SPA ON A 370/168 OR UNDER VM

In the System/370 Principles of Operation, the operation of the Common Segment bit (bit 30 of the segment table entry), is unspecified for those CPUs without the Extended Feature/Facility hardware. In most cases, including the 470 family, the bit is ignored. The 370/168, however, expects that bit to be zero, and causes a translation specification exception (interrupt code X'12') when a one-bit is found there. MVS/SEA was able to handle these interrupts by turning off these bits and re-dispatching the user. This occurs only twice, early in the IPL process.

VM, on a CPU without VM/SP, will not handle the common segment bits properly. No error message is given, and the translation specification exception mentioned may not occur, but unpredictable results, usually from wild branches, are seen. All remarks about the 370/168 also apply to systems running VM without VM/SP.

Due to the design of MVS/SPA, it is no longer possible to intercept the exception that is caused by these bits. MVS/SPA is routed control from the program FLIH only for operation or privileged operation exceptions, and thus cannot receive control on the translation specification exception. This will cause an 064-9 wait at IPL time. To avoid this problem on the 370/168 and under VM without VM/SP, it is necessary to superzap module IEAVNPX1, to prevent the common segment bits from being turned on. A change is made to IEAVNPX1 to prevent MVS from turning on the common segment bits if the system is running on a 370/168 or under VM. The bits will be turned on in all other cases. Please refer to AWS entry A#1208 for further details.

APPENDIX E - IMPLEMENTATION DIFFERENCES

MVS/SPA does not support all the features that the 370/Extended Facility and the 3033 Extension provide. The unsupported features are listed below, along with the reason they are not supported. A listing of deviations from the <u>System/370 Principles of Operation</u> follows.

E.1 UNSUPPORTED FEATURES

The following features are not supported by MVS/SPA:

- Suspend/Resume - Requires channel and CPU hardware changes.
- Subchannel Queueing - Requires channel hardware changes.
- LASP Instruction - Always simulated by other software on non-3033 Extension CPUs.
- ADDFRR Instruction - Always simulated by other software on non-3033 Extension CPUs.
- Low Address Protect - MVS/SPA supports this feature, but does not provide it. It is available as an EC to Amdahl 470 CPUs.
- Page Fault Assist - Requires CPU hardware changes.

E.2 DEVIATIONS

The deviations from the <u>System/370 Principles of Operation</u> are:

- The extraction authority and secondary space control bits of Control Register 0 (bits 4 and 5) are always assumed to be one. The bits are set on early in NIP and left on, so a software check would never fail.
- For READ and WRITE DIRECT, the operand address will always be treated as a logical address (subject to translation) on CPUs without the Extended Facility installed.

APPENDIX F – SCP MODIFICATIONS

MVS/SPA requires some superzap changes to the operating system in some environments. The installation instructions provide information on how to install these changes.

F.1 MVS/SPA UNDER VM OR ON A 370/168

The requirements are detailed in appendix D.

F.2 MVS/SPA ON A MACHINE WITHOUT 370/EF

Early in NIP processing, a check is made to see if the 370/EF feature is available by executing a TPROT instruction. If the instruction program checks, the system causes a WAIT 014. It is not possible for MVS/SPA to intercept this TPROT, so it must be NOP'd. The information on where the instruction resides and the proper offset to superzap is contained in AWS entry Z#1247.

F.3 MVS/SPA ON AN EXTENDED MEMORY 470 CPU

The following modifications are necessary to support the 470 Extended Memory features:

• IEAVNIPO	Ensures that the 470 EM feature is present and properly enables it.
• AMDSADMP	Three modules are changed to support the 470 implementation of the extended storage key instructions. MVS/SPA installation instructions for further details.

amdahl®

AMDAHL CORPORATION
1250 EAST ARQUES AVENUE
P.O. BOX 470
SUNNYVALE, CALIFORNIA 94086

READER COMMENT FORM

WE WOULD APPRECIATE YOUR COMMENTS AND SUGGESTIONS FOR IMPROVING THIS PUBLICATION.

Publication No.	Title	Current Date

How did you use this publication?
☐ Study ☐ Installation ☐ Sales
☐ Reference ☐ Maintenance ☐ Operations

What is your occupation?

What is your overall rating of this publication?
☐ Very Good ☐ Fair ☐ Very Poor
☐ Good ☐ Poor

Is the material presented effectively?
☐ Well organized ☐ Fully covered ☐ Clear
☐ Correct ☐ Well illustrated

Please enter your other comments below. If you were in any way dissatisfied with this publication, we would like to know why. Be specific, if possible; give page, column, and line number references where applicable.

All comments and suggestions become the property of Amdahl Corporation.

Your name & return address (include ZIP code):

AM2283 6/79

Thank you for your interest. Fold and fasten as shown on back. No postage necessary if mailed in U.S.A.

FOLD AND SCOTCH TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 741 SUNNYVALE, CA.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTENTION: TECHNICAL PUBLICATIONS
DEPARTMENT 734

amdahl®

AMDAHL CORPORATION
1250 EAST ARQUES AVENUE
P.O. BOX 470
SUNNYVALE, CALIFORNIA 94086

FOLD AND SCOTCH TAPE

AM2283 (Backer) 6/79