Using Cross-memory Services in MVS/ESA

James Antognini

IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, New York 10598

antognini@us.ibm.com

ABSTRACT

Cross-memory services are based on S/370 architecture and allow the user to have instruction and data access in more than one address space with various degrees of authority. In addition to the instructions provided by hardware (e.g., PC, PR, SSAR), MVS has services such as ETCON to establish and modify a cross-memory environment in which the instructions operate. This paper describes the general things one can accomplish, defines the basic instructions and MVS services, tells how to establish and exploit a cross-memory environment and how to observe that environment's restrictions, outlines debugging techniques and supplies some examples.

SCRIPTed on 20 January 2000 at 16:00:38

Acknowledgement

The original version of this paper was “Using Cross-memory Services in MVS/XA,” delivered by the author at SHARE 65, 1985, and published in the SHARE *Proceedings*. It is updated for MVS/ESA. The author gratefully acknowledges information and comments from Steven J. Greenspan, Peter J. Relson, Duna Williamson and Steven Kinder of Enterprise Systems and Nicholas S. Bowen of Research. Any mistakes are of course the author's responsibility.

Contents

Change History

Changes from the version published internally on 31 March 1993 are marked in the left margin.

8 September 1995 —

- ARR doesn't get control if entry table has been deleted and an error occurs.

24 September 1997 —

- Minor editing for publishing via FTP/the Web, e.g., copyright.

11 December 1997 —

- XEDITG figures replaced by figures from CorelDraw.

19 March 1999 —

- Description of problems and solution for system integrity when an SRB makes a specific connection of a PC-ss table.

1.0 Introduction

Cross-memory services appeared with MVS/System Product Version 1 Release 2 (MVS/SP1.2) and were significantly enhanced in SP3.1. The facility comprises a set of S/370 instructions and MVS services to manipulate hardware elements affecting those instructions. Even today understanding of the facility's workings is not common. The purpose of this paper is to lay out the fundamentals of the cross-memory facility.

The cross-memory facility makes several things possible:

Programs can be called with state changes, such as running in another address space, in supervisor state and in a different PSW key.

Data can be accessed in more than one address space and can be copied between address spaces.

Data can be accessed in more than one key, without the requirement of key-0 execution.

There are new kinds of authorization, more flexible than supervisor state and key 0.

Software can control the valid use and function of cross-memory instructions.

The enhancements to addressability, storage access by key, authorization and privileged instructions were accomplished in extensions of S/370 that were one of the more profound changes in mainframe architecture since the introduction of virtual addressing. Complementary to the S/370 changes were major enhancements to MVS, so that authorized programs can create and manipulate tables and values governing the function of cross-memory instructions. It is of course true also of virtual addressing that it depends on tables set up by software, but that mechanism is one restricted almost entirely to operating system components, whereas interfaces between authorized programs and the cross-memory facility have been integral to it since its inception.

There are several general purposes the cross-memory facility can serve. The first is to provide services in supervisor state or system key. Here “cross-memory” is a misnomer, since what is intended is simply execution in a more authorized state, without data access across address spaces. Routines relying on the cross-memory facility to execute in an authorized state have advantages over SVCs and programs in authorized libraries. One advantage is flexibility: Up to 2^{20} routines can be provided; they are not limited to enabled, unlocked, task-mode callers as SVCs are[1]; and they can be modified more readily than SVCs. The MVS service CPUTIMER is such a routine; it runs in supervisor state in order to issue the Store CPU Timer instruction to determine how much of a CPU interval remains. Another example might be an installation-written accounting routine to gather and save billing data in a system-key area of an address space, thus protecting the data from user inspection and tampering. A second advantage of the cross-memory facility in providing authorized-state routines is that the availability of such routines is easier to control than is that of SVCs and programs in authorized libraries. An illustrative case is GQSCAN, in which requests that might degrade system performance are limited to supervisor-state callers.

Another purpose of the cross-memory facility is data access, updating and copying in an address space and between address spaces. IMS, JES3, CICS and DB2, for example, do such things. IMS employs

---

[1] The ABEND SVC is the only exception, because the issuer intends to start the abend process. Other services are only apparently exceptions. ESTAEX, for example, is invoked by a PC call, not an SVC call.

cross-memory services to move data between its regions; DB2 copies data between its address spaces and application address spaces. An installation accounting routine like that already mentioned might use cross-memory services to move billing data collected in a user address space into another address space where the data would remain until processed by a periodic billing program. Isolation of data in an address space's private area renders them less susceptible to damage by failing routines in other address spaces and makes them readable only by the routines intended to access them.

A related purpose of the cross-memory facility is to invoke a routine to run in an address space other than the caller's. The routine executes synchronously, upon invocation, and so differs from the older mechanism of SRB routines that run at some point after they have been scheduled. The WTO/WTOR SVC, which may be issued in any address space, employs a cross-memory routine to carry out functions in the communications task address space. An installation accounting routine for job-step termination might in fact run in the accounting address space rather than the user's, particularly if the routine were to process the billing data rather than merely to copy them.

Performance might be another reason for utilizing the cross-memory facility in preference to SVC or SRB routines. SVCs have the overhead of the SVC interrupt handler upon invocation and of exit processing upon return, and SRBs have the cost of scheduling, dispatch and exit processing. Routines called by the PC instruction, on the other hand, get control directly, without software intervention, and return control directly, again without software intervention. As a result, a PC routine may have less overhead than an SVC or SRB routine.

Those are some general purposes of the cross-memory facility. Unfortunately, the facility, comprising as it does a wealth of S/370 instructions and operating system services, is very complex, difficult to use and perhaps daunting for first-time users. A fairly complete description of the MVS-provided services is contained in *MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs*, and of course *Enterprise Systems Architecture/390 Principles of Operation* lays out the instructions and underlying control structures. But these references don't answer every question that might arise, and they are of somewhat limited teaching value to a person who has never written a PC routine. *MVS/ESA Application Development Guide: Extended Addressability* provides a good introduction to the subject for that person.

The present discussion, in contrast to *Extended Addressability*, will be more concerned with hardware, especially with the role of control registers, indices and tables, and will develop the simpler uses of cross-memory services first. This paper is intended to show the experienced MVS systems programmer how to create cross-memory routines in an MVS/SP3 or SP4 system and to make *Authorized Services* and *Principles* more usable as reference works and *Extended Addressability* more valuable as background for topics like resource recovery.

1.1 Organization of the paper

The remainder of the paper is organized into these topics:

1. Definition of basic terms.
2. Overview of the cross-memory facility, including elements like linkage tables, macros, instructions and, most important, the sequence of steps necessary for an application to provide its own cross-memory services.
3. Programming examples for the steps.
4. PC routines, including their environment, programming examples and recovery.

5. Miscellaneous topics.
6. Debugging.

A reader somewhat familiar with cross-memory services may wish to concentrate on topics 3 and 4. Those less familiar should also read topic 1, at least the sequence of steps in topic 2, topic 6 and possibly topic 5. All of topic 2 is recommended to those wishing to understand the cross-memory facility from the perspectives of hardware and of MVS as a system control program.

2.0 Definitions

Whilst definitions of terms may seem tedious, the complexity of the cross-memory facility makes the definitions worthwhile, even though some of the terms won't be fully understandable until their particular applications are presented in later sections. General meanings are given here, and specific meanings will be amplified later as necessary.

Home address space is the dispatched address space, designated by PSAAOLD.

Primary address space is the address space whose segment table is used to fetch *instructions* when a program is executing in primary, secondary or AR ASC mode[2] (defined below) and to fetch or store data when in primary ASC mode. Upon initial dispatch, a task- or SRB-mode program executes in the primary=home address space. The primary address space may change as the result of PC, PR, PT or certain other instructions.

Secondary address space is the address space whose segment table may be used to fetch or store *data*. A program always has a secondary address space, and it is typically the same as the primary address space. The secondary address space affects program execution under specific conditions: The secondary address space has to be different from the primary, and, further, the program must use the MVCP or MVCS instructions or must be in secondary ASC mode or must use an access register containing the value 1 in AR ASC mode. At initial dispatch, a task- or SRB-mode program executes in the secondary=home address space. The secondary address space may change as the result of a PC, PR, PT, SSAR or certain other instructions.

Address space control (ASC) mode is indicated by PSW bits 16 and 17:

> ***Primary ASC mode*** is indicated by PSW bits 16 and 17 being off, so that instructions are fetched and data accessed by using the primary address space's segment table. That is, instructions and data are in the primary address space when a program is in primary ASC mode.
>
> ***Secondary ASC mode*** is indicated by PSW bit 16 being on and bit 17 being off, with the result that instructions are found in the primary address space and data in the secondary address space.
>
> ***Access-register (AR) ASC mode*** is indicated by PSW bit 16 being off and bit 17 being on. Instructions are found in the primary address space. Data, however, will be accessed by ALET-qualified addresses: A given datum will be in the address space, dataspace or hiperspace designated by the access register corresponding to the general-purpose register containing the datum's address.
>
> ***Home (AR) ASC mode*** is indicated by PSW bits 16 and 17 being on, so that instructions and data are found in the home address space. This mode is intended for operating-system components and will not be discussed further.

Address space number (ASN) or ***Address space ID (ASID)*** is the halfword value corresponding to an address space's ASCBASID and serving to designate the address space in many cross-memory services and instructions. ASN is employed in *Principles of Operation*; both ASID and ASN are used in MVS

---

[2] See Plambeck, "Concepts of Enterprise Systems Architecture/370," *IBM Systems Journal*, vol. 28, no. 1, 1989, page 45, for a brief history of ASC mode and instruction fetch and for hardware-related reasons for cross-memory enhancements.

manuals. HASN denotes the home address space, PASN the primary address space, and SASN the secondary.

Cross-memory mode, the only term in which "cross-memory" has a narrowly defined meaning, exists when at least one of the following is true of a program:

- The primary address space is not the home address space.
- The secondary address space is not the home address space.[3]
- Secondary ASC mode is active.[4]

CML lock is the local lock of an address space other than the home address space. (Every address space has an associated lock, namely, its local lock. The term "local lock" typically means the home address space's local lock.) The CML lock is *not* the CMS lock.

PC routine is a program invoked by the PC instruction. The routine will have been defined by MVS or by an authorized application like a subsystem and will generally get control with some kind of change in state. Routines that receive control without changing primary address space are referred to as PC-cp (current primary) routines, and those involving space switch are PC-ss routines. There is a further division of PC routines into basic and stacking, respectively corresponding to the original form introduced in SP1.2 and to the extended type that appeared in SP3.1 and that uses the linkage stack (described later). This paper will concern itself primarily with stacking-PC routines.

Cross-memory environment is a combination of tables, indices and control-register values that affect the execution of cross-memory instructions.

Cross-memory authorization is a set of mechanisms controlling use of certain cross-memory instructions, and it is not the same as APF authorization. (APF authorization is usually necessary at some point in setting up cross-memory services, since cross-memory macros require authorization either by the requester's PKM, defined below, or by virtue of execution in supervisor state or system key (0-7); and when a program isn't already in supervisor state or system key, it has to use the MODESET SVC, which *does* necessitate APF authorization.[5]) There are several varieties of cross-memory authorization:

> ***Program authorization*** is based on ***PSW Key Mask (PKM)***, a control-register value. The process provides for two distinct objects of key-related control over problem-state programs:
>
> - Calling a routine by the PC instruction. Such an invocation may be done by those problem-state programs whose PKM matches some part of the authorizing PKM in the definition of the PC routine.

---

[3] This and the previous bullet may be summarized as the condition where it is not true that PASN=SASN=HASN.

[4] Manuals such as *MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs* do not include secondary ASC mode in the definition of cross-memory mode, whilst *MVS/ESA Application Development Guide: Extended Addressability* does. The probable reason for the narrower definition in the *Development Reference* is that the some services (e.g., STORAGE) permit callers in AR ASC, PASN¬=HASN and SASN¬=HASN modes, but the services are invoked by a PC instruction, which cannot be effected in secondary ASC mode.

[5] APF authorization is fully developed in Antognini, "Writing Authorized Programs with System Integrity," SHARE 77 *Proceedings*.

- Issuing the instructions MVCP, MVCS, MVCK, MVCSK, MVCDK and SPKA (in-line expansion of the MODESET macro). These instructions involve a key that may be different from the PSW key, and they are permitted for problem-state programs whose PKM includes the specified key.

ASN authorization allows programs to issue certain PT, SSAR and PR instructions. The mechanism, explained in the next section, is enforced whether a program is in supervisor or problem state.

Owner, ***subsystem*** and ***service provider*** denote the address space that creates, alters and deletes cross-memory resources such as tables and indices. A service provider is not necessarily a classic MVS subsystem like JESx. The classic subsystem properties are, however, very useful to a service provider that owns cross-memory resources. In this paper's examples, a provider of cross-memory services is a classic subsystem.[6]

Local or ***dependent address space*** means an address space that uses cross-memory services provided by a service provider address space.

Tables are control blocks or structures. Instances are linkage tables, entry tables and authorization tables.

Indices are numbers, and commonly they designate positions in a table by being multiplied by a length factor to produce offsets into the table. Instances are linkage index, entry index and authorization index.

Cross-memory services are *not* narrowly defined in this paper and may refer to the facility in the S/370 architecture, to the services MVS provides via its cross-memory macros or to functions a service provider supplies via its particular application of the cross-memory facility in MVS.

---

[6] Consult Antognini, "Software Subsystems in MVS," SHARE 61 *Proceedings*, for particulars of classic MVS subsystems.

3.0 The Cross-memory Facility in Overview

3.1 The basic elements

The basic elements of a cross-memory environment are those resources that hardware examines to determine the legitimacy and particular function of cross-memory instructions. The resources take the form of tables, indices and control-register values. To understand how the MVS-supplied macros manipulate the environment, one should first understand its elements in a simplified form.

Entry table (ET)

Describes a collection of PC routines owned by a service provider. These routines actually supply the provider's cross-memory services. Each PC routine is described by an entry-table entry (ETE); the entry specifies the routine's entry point, whether the routine runs in supervisor state, whether space switch is involved, what is the authorizing PKM of a problem-state caller, what the PKM of the routine is to be, and so forth.

Entry index (EX)

Denotes a particular entry in an entry table by the entry's offset in the table.

Linkage table (LT)

Is the basis for connection of entry tables to an address space. One or more entry tables supplied by MVS are connected to an address space's linkage table at memory creation, and more entry tables may be connected by service providers at later times. Without connection of the table defining a set of PC routines, a program cannot even begin to invoke those PC routines.

Linkage index (LX)

Is the offset in an address space's linkage table for connection of an entry table. MVS-provided entry tables are connected at LXs reserved for the operating system. A number of so-called “system” LXs (55 by default, but the number can be changed by the NSYSLX parameter in SYS1.PARMLIB(IEASYSxx)) are reserved so that providers can make their services available in all address spaces by connecting entry tables to each linkage table at such LXs. For PC services that are to be available only in certain address spaces, the entry tables are connected to specific address spaces' linkage tables at the remaining LXs. A given linkage table will be connected to several entry tables, each connection at a different LX.

Authorization table (AT)[7]

Confers on programs the authority to make an address space the primary or secondary address space or to use an address space through an access list. First, when a program issues a PT-ss or SSAR-ss instruction (the target is not the current primary) or a PR-ss instruction (the new secondary is not the new primary), this table indicates whether the instruction will be carried out, for a PC-ss routine does not have inherent authority to issue PT-ss, SSAR-ss or PR-ss against its caller, even though the PC instruction may have made the caller's PASN the routine's SASN.[8] A second role of this table is to permit an address space to be placed onto an access list via an ALET or to be used through such a list, in conjunction with an EAX (explained below).

---

[7] *Principles* uses the term “authority” table.

[8] MVS will not allow the connection of the entry table of a PC-ss routine with its caller as its secondary address space if the relevant authorization table does not confer both primary and secondary ASN authorization.

Authorization index (AX)

Has two closely interrelated aspects. First, an AX is a control-register value inherited by a program from its primary address space. Second, an AX is an offset in an address space's authorization table: The address space confers on a program running in *another* address space with that AX value the authority to execute certain PT, SSAR and PR instructions that have the authorizing address space as target. In other words, the address space grants "primary" ASN authorization for PT and "secondary" ASN authorization for SSAR and PR.

Extended authorization index (EAX)[9]

Is, like AX, a control-register value utilized as an offset into an authorization table. The secondary ASN authorization bit at that offset indicates whether the authorization table's address space table can be added via an ALET to an access list in certain circumstances; the bit also says whether the target address space, if on an access list, can be used in AR ASC mode in certain circumstances.

PSW Key Mask (PKM)

Has two roles. The first is control over invocation of PC routines by problem-state programs: If there is no overlap between the caller's PKM and the required PKM, the PC call fails. The other function of PKM is to govern the storage keys to which problem-state programs have access via the MVCP, MVCS, MVCK, MVCDK and MVCSK instructions, and to control the PSW keys to which those programs have access by the SPKA instruction. PKM is not the same as PSW key, but since PKM can be employed to change PSW key via the SPKA instruction,[10] PSW key is effectively a subset of PKM so far as permitted values are concerned.

Linkage stack

An area associated with each dispatchable unit (that is, TCB or SRB). The status of a stacking-PC routine's caller (or of an issuer of the BAKR instruction) is contained in the current entry. This information comprises general-purpose and access registers, PSW, PASN, SASN, PKM and EAX, amongst other things, and is available to the PC routine for inspection and for restoring status at exit.

Control registers

Are part of the cross-memory environment. Instructions such as PC and PR and services such at AXSET may change control registers. These registers are not an intended interface, and changing or even inspecting them directly (for example, by LCTL or STCTL) is not recommended. The functions of the following control registers are, however, presented in brief for completeness:

1 Contains the real address of the primary address space's segment table.

3 Contains the PKM and SASN.

4 Contains the AX and PASN.

5 Contains a direct or indirect real-address pointer to the linkage table of the primary address space.

7 Contains the real address of the secondary address space's segment table.

8 Contains the EAX in bits 0-15.

13 Contains the real address of the home address space's segment table.

---

[9] Definition of EAX for a stacking-PC routine is one of two intended interfaces for getting a value into that control register. (SYNCHX is the other interface.)

[10] The in-line form of the MODESET macro generates a SPKA instruction.

15 Contains the address of the current linkage-stack entry and specifically of its descriptor field, which is located in the home address space.

The cross-memory environment is an integral part of program execution. Except for ownership of entry tables, LXs and AXs, every address space has the cross-memory resources listed above. Some resources may be held in common by address spaces; thus, address spaces with the default connections to MVS-provided entry tables and with the default primary and secondary ASN authorization share a system linkage table and a system authorization table. Furthermore, the default cross-memory environment includes an AX value of zero for the address space.

3.2 MVS services to manipulate the basic elements

MVS provides cross-memory macros manipulating the cross-memory environment. Most of these macros expand into calls of PC routines in the PCAUTH address space for actual service. The macros, to be examined in detail later, are:

ATSET
Confer primary and/or secondary ASN authorization to an address space.

AXEXT
Extract an address space's authorization index.

AXFRE
Free a previously reserved authorization index.

AXRES
Reserve an authorization index.

AXSET
Set an address space to an authorization index.

ETCRE
Create an entry table of PC routines.

ETCON
Connect an address space or all address spaces to a created entry table.

ETDEF
Define an entry table and its entries in private storage for later creation.

ETDES
Destroy a created entry table, possibly disconnecting address spaces from the table.

ETDIS
Disconnect an address space from an entry table.

LXFRE
Free a linkage index previously reserved.

LXRES
Reserve a linkage index.

PCLINK
Save status in a basic-PC routine. Not covered in this paper.

3.3 Cross-memory instructions

Cross-memory instructions depend on the cross-memory environment for execution. The instructions are:

PC
Program Call. Invoke a routine by its number.

PR
Program Return. Return control from a stacking-PC routine (or from a call by the BAKR instruction).

PT
Program Transfer. Return control, usually from a basic-PC routine (see section 6.6, “The PT instruction” on page 45 for a non-PC use).

IAC
Insert Address Space Control. Extract ASC mode.

SAC
Set Address Space Control. Set ASC mode.

SSAR
Set Secondary ASN. Designate a secondary address space by its id.

EPAR
Extract Primary ASN. Extract primary address space id.

ESAR
Extract Secondary ASN. Extract secondary address space id.

MVCP
Move to Primary. Copy data to the primary address space from the secondary.

MVCS
Move to Secondary. Copy data to the secondary address space from the primary.

SPKA
Set PSW Key from Address. Change PSW key.

MVCDK
Move with Destination Key. Copy data, with the access key affecting the target (PSW key governs the source).

MVCK
Move with Key. Copy data, with the access key affecting the source (PSW key governs the target).

MVCSK
Move with Source Key. Copy data, with the access key affecting the source (PSW key governs the target).

EREG
Extract Stacked Registers. Obtain the contents of one or more general-purpose and access registers from the current linkage-stack entry.

ESTA
Extract Stacked State. Obtain information such as caller's PSW from the current linkage-stack entry.

MSTA

Modify Stacked State. Change the 8-byte modifiable area in the current linkage-stack entry.

LASP

Load Address Space Parameters. Load values into control registers. Used by MVS routines to switch cross-memory environments. Not an intended interface and not covered further in this paper.

The BAKR instruction and a PR instruction used to reverse the effect of a BAKR are not considered cross-memory instructions here, even though they affect the linkage stack.

3.4 Extended addressability via AR ASC mode

Extended addressability through AR ASC mode is historically distinct from cross-memory services, since basic-PC services appeared first.[11] Furthermore, AR ASC mode can be used by programs that make no use of PC routines. Yet PC routines will often have to take AR ASC mode into account—a routine may, for example, allow callers to specify ALET-qualified addresses—and for that reason the ESA extensions of the cross-memory facility accommodate AR ASC mode.

It may be instructive to list some differences between cross-memory concepts and AR ASC mode concepts:

1. Cross-memory concepts apply to address spaces, and PASN, SASN and HASN designate only address spaces. AR ASC mode extends to dataspaces and hiperspaces as well.
2. Cross-memory mode isn't AR ASC mode.
3. Cross-memory services typically involve state changes, such as those effected by a PC instruction to a new PASN or SASN, to supervisor state, and so forth. AR ASC mode does not require a change, and the mode enables a program to enjoy persistent addressability to address spaces, dataspaces and hiperspaces.
4. Cross-memory services are built on linkage tables, entry tables, authorization tables, linkage indices, entry-table indices and authorization indices. AR ASC mode is indicated in the PSW and is supported by access registers, ALETs and access lists.

There are areas where cross-memory services and AR ASC mode overlap:

1. Both may be employed for addressability to other address spaces, by change (via a PC instruction) or by persistent addressability (via AR ASC mode)
2. The secondary address space can be accessed in AR ASC mode by the ALET value of 1, because that value is always effectively on the access list (as is the ALET value of 0, which allows access to the primary address space). Consequently, AR ASC mode renders secondary ASC mode and the MVCP and MVCS instructions redundant and effectively obsolete.
3. EAX is most readily employed by a combination of cross-memory services with AR ASC mode, namely, in a PC routine using access registers.
4. When a stacking-PC routine is established by MVS cross-memory macros, its definition includes specification of whether it is to get control in primary or AR ASC mode (independently of the caller's ASC mode).

---

[11] See Clark, "The facilities and evolution of MVS/ESA," *IBM Systems Journal*, vol. 28, no. 1, 1989, for a overview of cross-memory and extended addressability.

5. The PT, MVCP and MVCS instructions are disallowed in AR ASC mode, as is a PC instruction to invoke a basic-PC routine.
6. ALETs of 0 and 1 as parameters change meaning if a PC-ss routine is called, since those values will mean different things for the caller and the routine.

3.5 Outline of providing cross-memory services

3.5.1 Linkage and entry tables

The following figure represents a particular cross-memory environment telling hardware how to execute a PC instruction. An address space's linkage table has been connected to entry tables of two different service providers, at offsets LX=9 and LX=11 (connections to operating system entry tables are not shown). A program in the address space desires to call PC routine 2 of the first service provider (“subsystem 1” below), so the fullword value X‘00000B02’ (LX=11 + EX=2) is specified in a PC instruction. To accomplish the call, hardware carries out a two-table look-up process with that PC number, as follows:

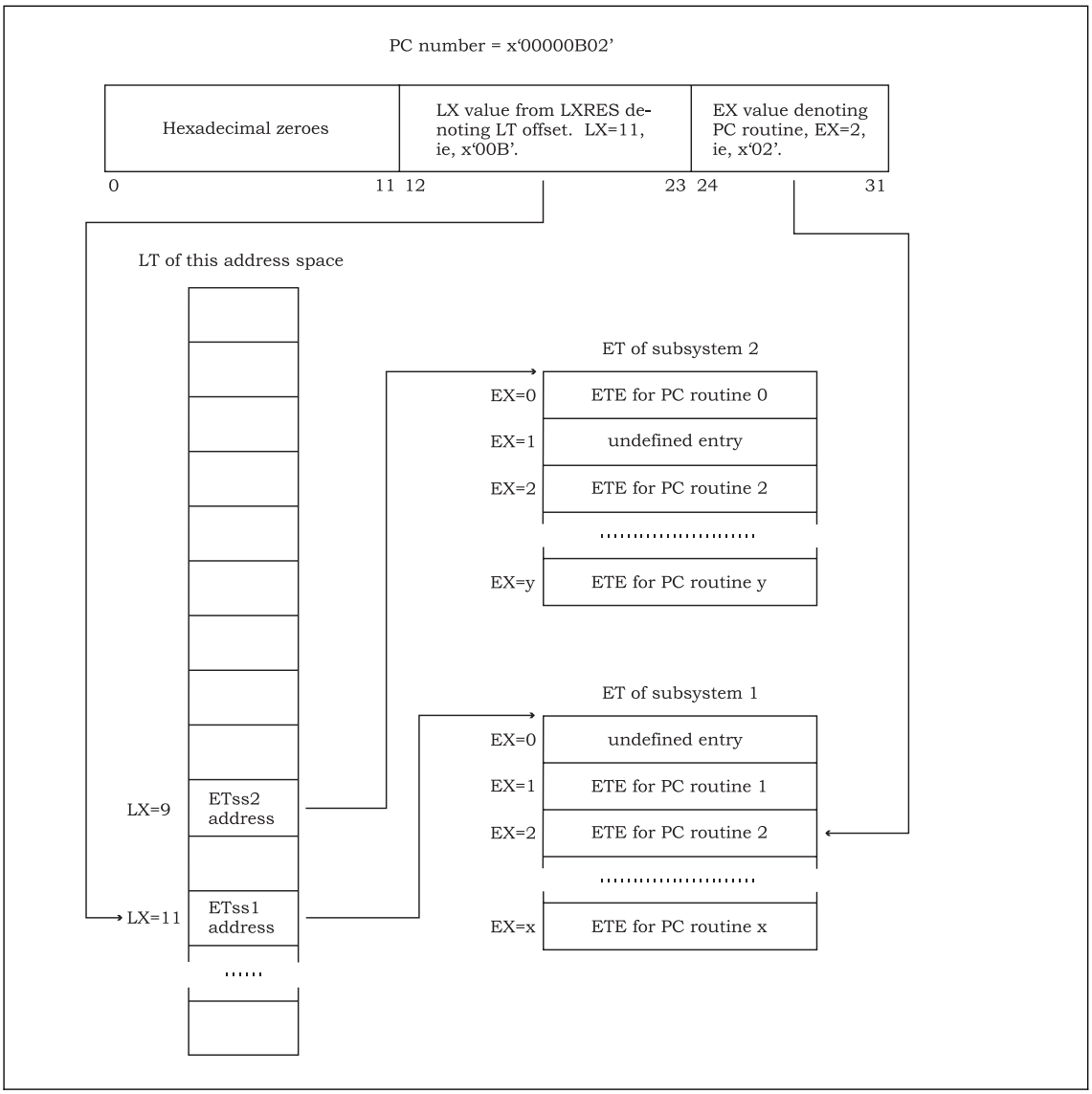

Figure 1. Entry table linkage and the role of a PC number in effecting a PC-routine call

3.5.2 Authorization tables

The authorization process for certain PT, SSAR and PR instructions and sometimes for instructions in AR ASC mode is distinct from the mechanism controlling PC instructions. Authorization is required for changing the primary address space by PT-ss (the PASN-to-be is different from the current PASN), the secondary address space by SSAR-ss (the SASN-to-be is different from the current PASN) and the secondary address space by PR-ss (the SASN-to-be is different from the PASN-to-be).[12] Primary ASN authorization is enforced for such PTs, and secondary ASN authorization, for such SSARs and PRs and, in certain cases, for operations in AR ASC mode.

[12] Whilst a PR to return from a stacking-PC routine can change PASN and SASN, a PR to return from a BAKR instruction cannot change either. Hence this authorization checking is not done for BAKR.

An entry in an authorization table consists of two bits. The first bit in an entry stands for authority of a program with the corresponding AX to establish the address space as primary, and the second bit, for authority to establish the address space as secondary. For checking PT and SSAR instructions, the AX value in force at the point of the instruction is used; for checking a PR instruction, the AX value of the new primary address space is used. The bit for secondary ASN authorization has a further role: It may be checked to determine whether a program with the corresponding EAX can execute instructions in AR ASC mode against the address space, for example to access the address space's storage. (See section 6.3, “EAX” on page 42, on how an EAX is used.)

The next figure depicts a particular address space's authorization table. The entries for AX=0 and AX=1 were set at address space creation and cannot be changed. The bits in the AX=0 entry mean that programs whose primary address spaces have AX=0 cannot issue PT to make the “target” primary (when the address space is not already the primary) and cannot issue SSAR or PR to make that target secondary (when the address space is not already the primary or the new primary, respectively). The bits in the AX=1 entry mean that programs whose primary address spaces have AX=1 can do all those things.

The address space in this example has granted primary and secondary ASN authorization to programs with AX=6 or AX=7 and only secondary ASN authorization to programs with AX=4. The AX=6 entry was probably set to accommodate PC-ss routines with their callers as secondary, and the AX=4 entry was probably intended for PC-cp routines accessing the address space in AR ASC mode.

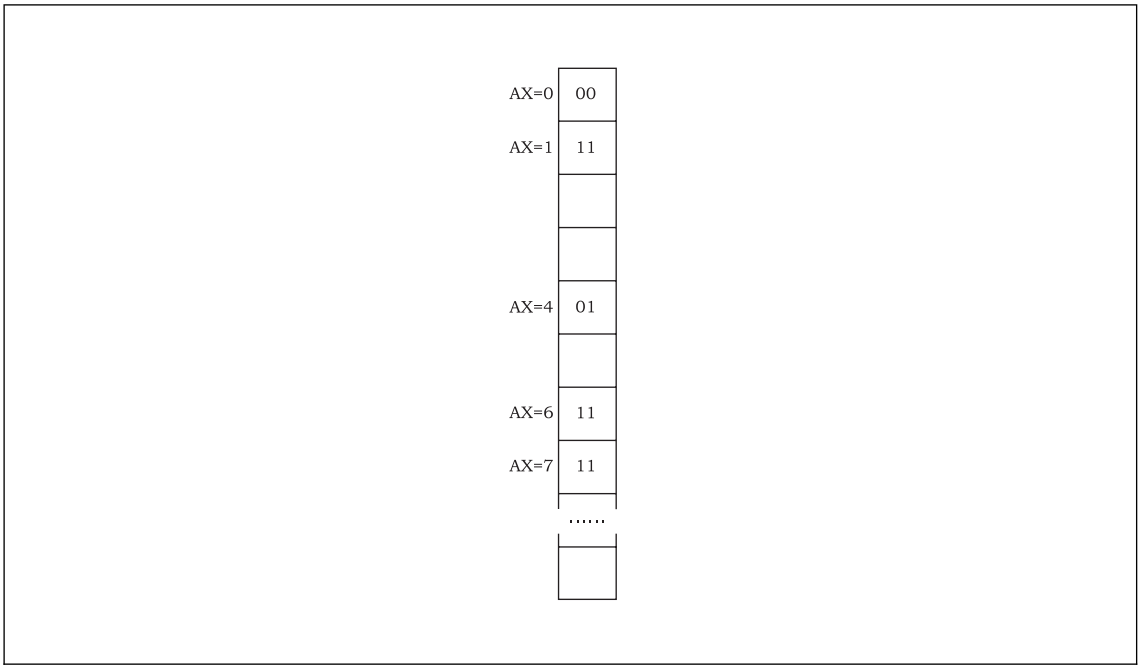

Figure 2. Authorization table of an address space

3.5.3 The steps in offering and withdrawing cross-memory services

MVS provides services to establish and manipulate the linkage and authorization components of the cross-memory environment. The figures above illustrate the end result of those services: PC routines are callable in an address space because, and only because, the entry table describing those routines has been connected to the linkage table of the calling address space. Further, if any of the routines are PC-ss for which the caller's primary address space becomes the PC routine's secondary, or if a PC routine wishes to use an address space through an access list, the calling address space's authorization table has a role in offering cross-memory services in PC routines.

The fundamental requirement for supplying cross-memory services is connection of an entry table to the linkage table of an address space. The steps to achieve that condition and those to end that condition are indisputably complex, but if one keeps in mind that they all lead up to or away from connection, comprehending the logic behind the steps is much easier. The next two figures are diagrams of the steps before and after connection. A description of the steps will immediately follow, as a second-level explanation. Finally, in section 4.0, “Cross-memory Macros and the Services They Invoke” on page 20, detailed examples will be a third-level explication of the steps a service provider would follow to offer its own cross-memory services through the PC instruction.

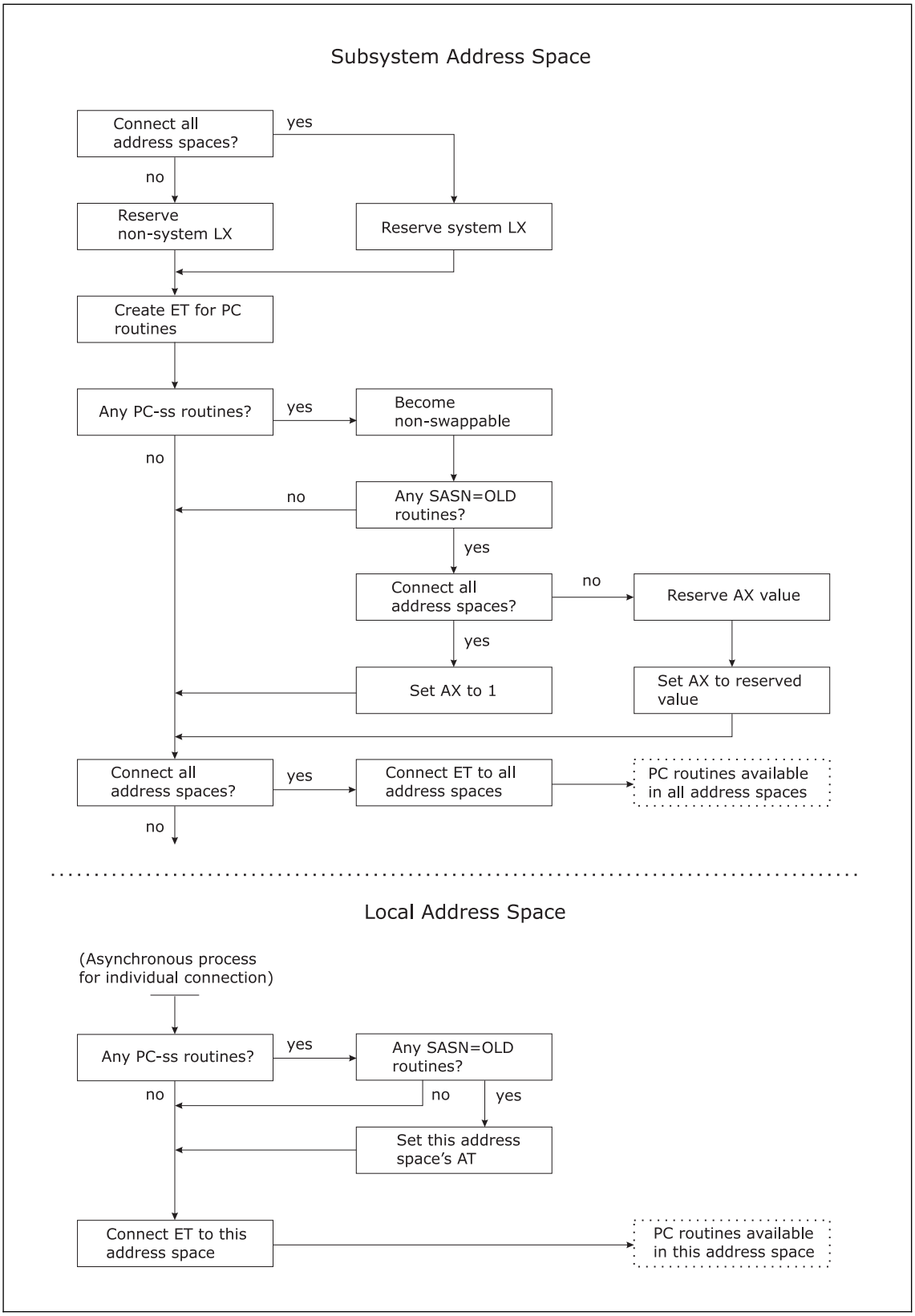

Figure 3. Initialization of a service provider's cross-memory services

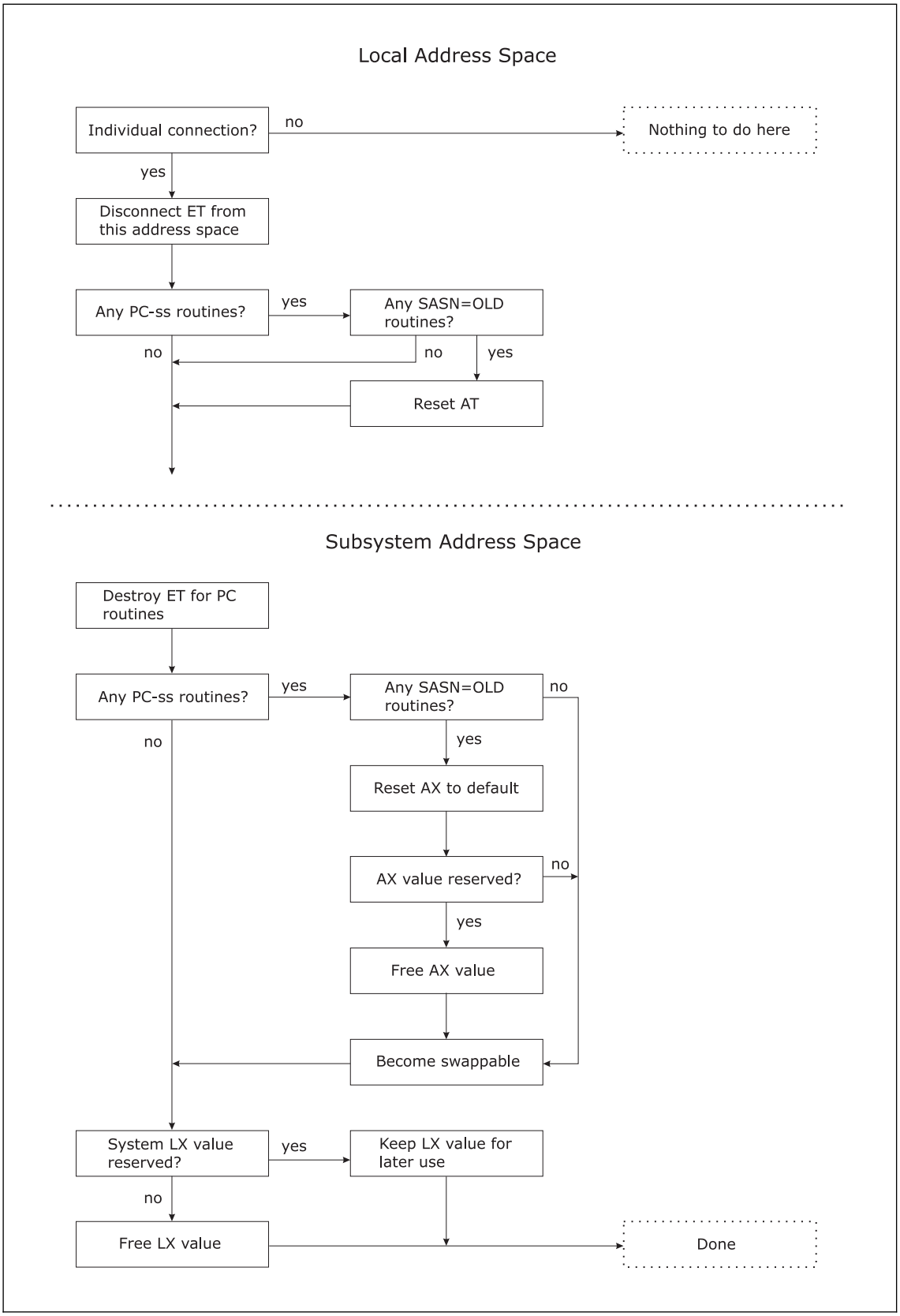

Figure 4. Termination of a service provider's cross-memory services

Note: The steps for using an EAX are not shown. See section 6.3, “EAX” on page 42, for that information.

These are the steps for a service provider to offer cross-memory services via PC routines:

1. An entry is reserved by the service provider in the linkage table of every address space, via the LXRES service. If this entry is to be used to connect every address space and not merely certain ones, the reserved LX must be a system LX. The service provider address space owns the reserved LX.
2. The service provider creates an entry table describing its cross-memory services, via ETCRE. The service provider address space is the owner of the entry table.
3. If the service provider is to offer PC-ss routines, one or more steps are required:
   a. The service provider address space makes itself non-swappable, since a space-switching PC call to a swapped-out address space would result in an abend.

      If the entry table has no entries where the caller's primary address space is to become the PC routine's secondary (that is, no SASN=OLD entries, as explained later), steps 3b through 3d are skipped.

   b. If the entry table will be connected only to certain address spaces, the service provider may elect to reserve an entry in every address space's authorization table, via the AXRES service. The reserved entry, designated by its AX offset, is owned by the service provider address space. (AX reservation is unnecessary if the service provider will be connecting the entry table to all address spaces, since an AX of 1 is to be used in such a case and that value does not demand reservation.)

   c. Since PC-ss routines will run in the service provider address space, the service provider sets its AX value, via AXSET. An AX value of 1 can be instated, giving any program running in the service provider address space primary and secondary ASN authorization to any address space; step 3d is then skipped. Alternatively, a reserved AX value can be instated, giving programs in the service provider address space authorization to only those address spaces that will have permitted PT-ss, SSAR-ss and PR-ss by programs with that AX.

   d. If the service provider address space sets its AX to a reserved value, a local address space, before connecting an entry table for PC-ss routines that are to have the local address space as their secondary, must confer primary and secondary ASN authorization on programs running in an address space with that AX; authority is granted via ATSET.
4. To make services actually available in an address space, the entry table must be connected to that address space's linkage table at the entry designated by the reserved LX. Connection is effected via ETCON: If connection is to apply to every address space, the service provider itself requests the connection, utilizing its system LX; if, on the other hand, connection only in certain address spaces is desired, it must be requested in each of those address spaces.
5. After connection, the service provider's cross-memory services can be invoked via PC instructions issued in the address space. The fullword operand in a PC instruction identifies an entry table by its LX offset and a routine in that entry table by its EX offset.

When the service provider's PC services would no longer be invoked, the service provider reverses the steps above, undoing connections, primary and secondary ASN authorization, non-swappability and so forth as necessary. These, briefly, are the steps:

1. If the local address space was individually connected to the entry table,

a. The entry table is disconnected from the address space's linkage table by ETDIS in the subject address space.
b. The address space's authorization table is reset via ATSET if it was set before connection.

2. If the entry table was connected separately to address spaces, it is destroyed via ETDES in the service provider address space once all connections have been broken. If the table was connected to all linkage tables, it is disconnected and destroyed in the service provider address space by ETDES with purging.

3. If the entry table had PC-ss routines,

   a. If any PC routine was defined with SASN=OLD,
      1) The service provider address space's AX is reset by AXSET.
      2) Any reserved AX value is freed by AXFRE.
   b. The address space is made swappable.

4. If a system LX was reserved, its value is kept for reuse. If a non-system LX was reserved, it is freed by LXFRE.

4.0 Cross-memory Macros and the Services They Invoke

4.1 Scenarios

Three broad applications of cross-memory services will be the basis for specific examples in this section.

A routine callable in any address space, to run in the caller address space, with more authority than the caller has.

A routine callable in any address space, to run in the service provider's address space, with more authority than the caller has.

A routine callable in designated address spaces only, which runs in the caller or the service provider address space, with more authority than the caller has.

Callable from anywhere, executing in the caller address space, with more authority

There has always been a need for a way to call routines that have more authority than the caller and that almost any program can call legitimately. Historically, SVCs have served this function, but the SVC approach has some drawbacks: There is a limit to the number of SVCs that can be defined, and changing SVC modules for reasons of maintenance and adding SVCs for new functions are cumbersome processes. (A comprehensive comparison of PC routines and SVC routines is provided in section 6.1, “PC routines versus SVC routines” on page 40.) Cross-memory services make possible routines invoked by the PC instruction, and PC routines in this first category would have some of the following attributes:

- The routines are callable in any address space, because the owning address space, the service provider, has made them callable from anywhere.
- Although callable in any address space, the routines might be callable by problem-state programs only if they were running in PSW keys designated by the owner.
- The routines will execute in the caller address space, and they are PC-cp routines.
- These routines will typically have greater authority than does the caller. That authority can take the form of execution in supervisor state, execution in PSW key 0-7 and access to more PSW and storage keys. By having this authority, the routines have available most MVS services and specifically can use instructions that are restricted to supervisor state and can change PSW key or access and perhaps copy data from one key to another.

Callable from anywhere, executing in the service provider address space, with more authority

These routines are like those above, except that:

- They run in the address space of the service provider, rather than in that of the caller; that is, the service provider address space becomes the primary. The routines are PC-ss routines.
- The primary address space at the time of the PC instruction may become the secondary.
- The routines can readily copy data between the caller and service provider address spaces if the caller address space has become the secondary.

Although there is a space switch, the routines run with the caller's dispatchability, that is, with the dispatching priority of the home address space and of the unit of work.

Callable from selected address spaces, possibly executing in the service provider address space, with more authority

These routines share many attributes with the first two categories but possess this distinguishing characteristic: The service provider, when employing cross-memory macros to instate the routines, has not made them generally available. Rather, in each address space needing these routines, some program will have employed MVS cross-memory services to make the routines available.

Since calling MVS's cross-memory services requires supervisor state or PKM 0-7 and since programs needing the services may not be able to assume that authorized state, the service provider must provide some indirect way to obtain the services. A typical technique would be a “bridge routine” in the form of a PC-cp routine, callable from anywhere, which would perform some checking to ensure the legitimacy of the request and would then make the restricted routines available.

4.2 General characteristics of the service provider

Setting up a provider's cross-memory services is done by the owner of those services, the service provider. The service provider will have the following attributes:

- The service provider must be APF-authorized, because the MVS cross-memory services to instate a cross-memory environment for address spaces will require at least supervisor state or PKM 0-7, and these latter are obtainable by the service provider only if it starts life APF-authorized. Thus, a service provider would be an authorized program from an authorized library and so would have job-step authorization. (An alternative is for the service provider's job-step program to be a Program Properties Table program that is from an authorized library and that gets control with PSW key 0-7.)
- The service provider must persist as long as its services are needed and are being exercised, since termination of its services could have undesirable effects on programs using the services. The need for uninterrupted service imposes a requirement on the service provider for recovery from errors, lest they cause termination of its job step.
- The service provider should provide for its own clean up of resources when it is about to terminate, since the operating system's clean up at job-step termination may be different from or more drastic than what the service provider and its dependents could tolerate.
- If the service provider provides PC-ss routines, the service provider address space must ensure that it will not be swapped out, since a PC call of a routine in a swapped-out address space would result in a program check and hence an abend (the same applies to PT-ss, SSAR-ss and PR instructions[13]).

[13] The reader may think that since a program can execute only when its home address space is swapped in, PT-ss, SSAR-ss and PR-ss are guaranteed not to fail. This is not the case. PT-ss, for example, can be used against any address space, so long as there is primary ASN authorization, and not just against the home address space; SSAR-ss likewise can be directed against any address space. PR of course is used for return, but if a PC-ss routine in one service provider's address space has invoked a PC-ss routine in another service provider's address

- The service provider address space is similar to address spaces like PCAUTH, in that the provider's cross-memory services when invoked are effectively subroutines of the caller, even when those routines execute in the service provider address space. Thus, a service executes with the caller's dispatchability and is charged to the caller's home address space and dispatchable unit.
- The service provider, in this paper's examples, will be a classic subsystem à la JESx, for the control block chains associated with such a subsystem make it easy to locate certain subsystem information from any address space.

The service provider does not manipulate the cross-memory environment directly, for a great deal of it actually resides in the private area of the PCAUTH address space and, still more importantly, because that environment should be manipulated in a proper fashion, with provisions for system integrity, legitimacy, correctness and serialization. Macros have been supplied for the MVS services that actually change the cross-memory environment. These cross-memory macros have a number of restrictions on the execution states in which they may be used. When issuing the macros, service provider programs are enabled, unlocked and able to address any parameter lists in primary ASC mode, in addition to running in supervisor state or with PKM 0-7.

4.3 The steps for generally connected PC-cp routines

The most straightforward application of cross-memory services is a PC routine that is callable from any address space and that is to run in the caller address space. The substantive function of the routine might be to provide for the caller some MVS services requiring execution in supervisor state or system key; no transfer of control or movement of data between the caller and the service provider is involved or, in fact, provided. The PC routine in this first example might, for example, be invoked to write an SMF record that an installation needs for its accounting.

These are the steps in providing such a PC routine and in a program's use of the routine.

1. The service provider address space is already running in supervisor state following initialization. Supervisor state is useful because many cross-memory macros require supervisor state or PKM 0-7, and supervisor state makes PKM checking unnecessary and hence is a bit more efficient. Furthermore, supervisor state permits use of in-line MODESET to change keys easily and efficiently via the SPKA instruction; the service provider might generally run in key 8 and switch to key 0 only for those operations requiring that key, because keeping key-0 execution to a minimum reduces potential damage from software errors.
2. The service provider should set up a recovery environment by ESTAE or equivalent service. The service provider's job-step task will be the owner of cross-memory resources—entry tables, LXs and AXs—created or reserved in that address space, and it is generally advisable to do one's own clean up of resources rather than to rely upon end-of-task or end-of-memory clean up by MVS.
3. The service provider gets a system LX, which it will need in order to offer services in all address spaces:

space, PR in the second routine will not be returning to home but rather to the first service provider's address space; hence, the PR will work only if that address space is swapped in.

```
* DSECTs mapping fields like LXCOUNT as well as operating system con-
* trol blocks appear in section A.2, "Mapping DSECTs" on page 51.
        LA    R1,1              load and save 1
        ST    R1,LXCOUNT          as number of LXs needed
        LXRES LXLIST=LXL,       get a system LX, with its value      +
              SYSTEM=YES,         returned in LXVALUE                +
              MF=(E,LXRESLST)
```

(Note that the count field and the returned values are fullwords.)

The service provider's job-step task (the cross-memory ownership task[14]) owns the reserved LX, whether reservation was done by that task or by a subtask (or even by an SRB routine in the address space). Whatever process reserves the LX should save the returned LX value in a place where it can be found again if the service provider is brought down and later restarted. Saving the LX value permits the service provider to reuse this LX, and system LXs should be reused since there is a limited number available in the life of an IPL. If the service provider is a classic MVS subsystem, it could save the LX in an extension of its SSVT. Such an SSVT *cum* extension, if obtained in a subpool like 241, would be readily located in any address space and would persist beyond the termination of the service provider address space. Examples in this paper assume an extension of the subsystem's SSVT when there is need for commonly addressable and persistent areas.[15]

```
* The service provider, in initialization, obtained enough storage in
* subpool 241 (CSA storage persisting until explicit freeing) in key
* 0 to create a full SSVT and an extension.
* At this point, the service provider is running in supervisor state
* and has saved its SSVT address in SAVESSVT and its usual PSW key
* (e.g., key 8) in SAVEKEY.
* HRLMSSVT (see page 51) maps the subsystem's extension to the SSVT.
        L     R1,SAVESSVT       point to HRLM's SSVT
        MODESET EXTKEY=ZERO     get key 0
        USING SSVT,R1
        MVC   HSSVTLX2,LXVALUE  save returned LX in SSVT extension
        DROP  R1
        MODESET KEYADDR=SAVEKEY, back to usual key                   +
              WORKREG=1
```

(With the advent of MVS/SP4.2.2, name-token callable services are an alternative means for storing and retrieving things such as the LX. But even where those services are available, use of an SSVT extension may be preferred because retrieval from it is probably more efficient than retrieval by a name-token service.)

4. Routines to be called via PC instructions must be available, and since they will be callable in any address space without a switch to the service provider address space, the modules must reside in common storage. Possible sites are the nucleus and the LPA. An alternative is to load programs into CSA or SQA storage, and this may be accomplished by the “load to global” form of LOAD (i.e., LOAD with GLOBAL) or by the “directed” form of LOAD (i.e., LOAD with ADDR) for which storage has already been obtained.

---

[14] The task whose TCB is denoted by ASCBXTCB, namely, the task under which the program named in JCL's EXEC=*program* got control.

[15] The SSVT should probably be locatable via the SSCTSUSE (or the SSCTSUS2) field as well as the SSCTSSVT field. For, whilst the subsystem should zero the latter when it is terminating so that the operating system will recognize the subsystem as inactive, the SSVT's storage need not be freed. If the SSCTSUSE were then left pointing to the SSVT, the service provider in restart could readily find the SSVT and its extension.

Directed LOAD is handy, because storage can be obtained in a subpool that won't be freed at task or memory termination, and because storage in the CSA can be obtained on a page boundary (aiding debugging). An example is given in appendix A.1, “Loading a PC routine” on page 50. If LOAD to global is used, the EOM=YES parameter may be used to prevent deletion before the owning address space terminates. For the service provider must ensure that the modules are not deleted before it intends to withdraw its cross-memory services; specifically, modules should not be loaded under a subtask if end-of-task processing would delete them before the associated service provider services are supposed to end. (Resources acquired through cross-memory macros such as LXRES are owned by the job-step task and thus are not subject to the same degree of unintended deletion or reclamation by the operating system.)

5. An entry table description list for the PC routines is prepared. There are several things that must or may be put into this description:

   a. The number of entries must be specified.

   b. An entry for a PC routine must be designated as basic or stacking. Certain options are restricted to stacking-PC routines, and only such routines will be considered hereafter.

   c. The EX must be set in the entry, and entries must be ordered by ascending EX value, although the numbers need not be consecutive. Examples here use EXs from 0 to *n*. EXs with consecutive ascending values are automatic if a series of ETDEF TYPE=ENTRY macros is employed for definitions in a CSECT. (But see the note about EX values on page 26.)

   d. The Authorization Key Mask (AKM) for checking a problem-state caller's authority to invoke the routine is set in the entry. The AKM is the string of keys in which the routine may be called: This value will be ANDed with the caller's PKM, and the call will proceed if a non-zero value results; otherwise, an 0C2 abend will ensue.

      Note: AKM and PKM are distinct entities.

   e. The name or entry point of the routine must be given in the entry. If a routine is designated by its name, the 8-byte name (padded with blanks) is used, and the routine must reside in the fixed, modified or pageable LPA or in the nucleus; this technique is not shown here. If a routine is designated by entry point, the 4-byte address is used, as shown below.

   f. Various state attributes for the routine may be specified:

      1) The entry may be set so that the associated routine will receive control in supervisor state.
      2) The entry may be set so that the routine will run in the service provider address space and not in the caller.
      3) The entry may be set to indicate whether the routine's secondary address space will be the caller's primary (SASN=OLD, the default) or the new primary (SASN=NEW). Specification of the secondary is important only if the routine is to run in the service provider address space, since if the routine is to run in the caller address space, the secondary address space of the routine will always be the caller's primary address space.
      4) The Entry Key Mask (EKM) may be set in the entry. This bit string either will become the routine's PKM or will be ORed with the caller's PKM to form the PKM.
      5) If the routine's entry point is specified, addressing mode may be set as 24- or 31-bit.
      6) Two 4-byte parameters are set in the entry. These values are referred to as the latent parameter, and a copy of them will be addressable at entry to the PC routine. The latent parameter values might address areas in common storage. If no parameters are specified, the latent parameter will consist of two zeroed fullwords. The definition in ETDEF1 below shows the setting of the second latent parameter value. Its use is shown in point 5 on page 35.

7) The ASC mode may be specified as primary or AR.
8) The EAX may be specified. If specified, it must be an AX value owned by the service provider address space. If none is specified, the PC routine will have the EAX of its caller.
9) The routine's PSW key upon getting control may be specified. If none is specified, the PSW key will be the caller's.

g. An associated recovery routine (ARR) may be specified. This is like an ESTAE routine in force at entry to the routine. ARRs are discussed in section 5.4.1, “Associated recovery routines (ARRs)” on page 38.

The example below uses ETDEF to build a description list (note that the default with ETDEF is a stacking-PC definition). The example is taken from prototype code that does eight ETDEFs in a loop. (Code to load the PC routines is shown in appendix A.1, “Loading a PC routine” on page 50.)

(This example and indeed all examples assume that working storage has been initialized to zero, so there is usually no setting of fields to zero values.)

```
* Copy static definitions to working storage.
         LA    R14,ETDLIST        target address
         LA    R0,ETDLBASE        source address
         LA    R15,ETDLISTL       target length
         LR    R1,R15             source length
         MVCL  R14,R0             copy static entries to working store
         ...
         LA    R9,1               initialize loop index
ETDEFLP  DS    0H                 R9 = 1 to 8
         ...                      load a PC routine
* R2->PC routine loaded into commonly addressable storage
         LR    R3,R9              copy loop index
         BCTR  R3,0               now R3 = 0-7
         LA    R1,ETD1-ETD0       get length of an entry
         MR    R0,R3              get offset to current entry
         LR    R15,R3             copy index - 1
         LA    R3,ETDLIST+(ETD0-ETDLBASE)(R1) get address of entry
         SLL   R15,2              multiply index - 1 by 4
         B     ETDEFTBL(R15)      branch to wherever
ETDEFTBL DS    0H
ETDEFIX0 B     ETDEF0             index=0:  define type 0
ETDEFIX1 B     ETDEF1             index=1:  define type 1
         ...
ETDEFIX7 B     ETDEF7             index=7:  define type 7
* At each ETDEF, R3->entry in description list and R2->routine's entry
* point.  Each entry is modified.
ETDEF0   ETDEF TYPE=SET,          PC-cp routine                        +
               ETEADR=(R3),         this entry element                +
               ROUTINE=(R2),        this entry point                  +
               AKM=(0:15),          if problem state, any PKM OK      +
               EKM=(0:15),          all values                        +
               PKM=REPLACE,         EKM values replace caller's PKM   +
               SASN=OLD,            SASN is caller's PASN (default)   +
               STATE=SUPERVISOR     supervisor state
         B     BUMPCNT
ETDEF1   DS    0H                                                     +
* Set second latent parm value (first defaults to 0).
         L     R6,=X'12345678'    get value for 2d half of latent parm
         ETDEF TYPE=SET,          PC-cp routine                        +
               ETEADR=(R3),         this entry element                +
               ROUTINE=(R2),        this entry point                  +
```

```
                AKM=(0:7),          if problem state, only PKM 0-7 OK  +
                EKM=(0:15),         all values                         +
                PKM=REPLACE,        EKM values replace caller's PKM    +
                EK=0,               in PSW key 0                       +
                SASN=OLD,           SASN is caller's PASN (default)    +
                PARM2=(R6),         set second latent parm value       +
                STATE=PROBLEM       problem state
        B       BUMPCNT
        ...
BUMPCNT DS      0H
* Loop back unless number of PC routines exceeded (see appendix
* A.1, "Loading a PC routine" on page 50, for the list of routines).
        LA      R9,1(,R9)           bump loop index
        CH      R9,=Y(PCRTNNBR)     compare to number of PC routines
        BNH     ETDEFLP             go back unless that number exceeded
        ...
ETDLBASE ETDEF TYPE=INITIAL         table of entry descriptions
ETD0    ETDEF TYPE=ENTRY,AKM=0,ROUTINE=0
ETD1    ETDEF TYPE=ENTRY,AKM=0,ROUTINE=0
ETD2    ETDEF TYPE=ENTRY,AKM=0,ROUTINE=0
        ...
ETD7    ETDEF TYPE=ENTRY,AKM=0,ROUTINE=0
        ETDEF TYPE=FINAL
ETDLISTL EQU   *-ETDLBASE
```

Note: The ETDEF TYPE=SET macro does not set the EX value. If the value needs to be set (for example, to use a non-consecutive number), the value can be set in the one-byte field ETDEX by using an MVI instruction along with the IHAETD mapping macro.

6. The service provider invokes the ETCRE service to create an entry table from the description list; the token returned by ETCRE must be saved, since it will represent the table in a later connection:

```
        ETCRE ENTRIES=ETDLIST
        ...                 check return code
        ST    R0,TKVALUE    save token
```

   (Note that the returned token is a fullword.)

   The service provider's job-step task owns the entry table, whether creation was done under that task or under another dispatchable unit (TCB or SRB) in the address space.

7. PC numbers for calling the routines have to be constructed by the service provider or otherwise made available.

   A PC number is a fullword in which bits 12-23 denote an LX and bits 24-31 denote an EX; hardware will execute a PC instruction by using the LX value to locate the entry table pointer in the linkage table and the EX value to locate the entry in the entry table. The service provider may build PC numbers for calling the routines, inasmuch as the LX part of a number is not known until the LXRES service returns the value. PC numbers can be formed by adding the fullword form of each routine's EX to the LX, since LXRES returns the value in bits 12-23 of a fullword for just this purpose. Once built, PC numbers would be made known to any address space where a program would call the PC routines. There is, however, no convention for the format and location of the PC number table, so the service provider and its dependents observe their own convention.

   An alternative technique, used in this paper, is to make only the LX available in a table appended to the subsystem's SSVT (see point 3 on page 22). This value can be found in any address space by following a short chain of system control blocks (see point 9 below), and a PC number can be formed from the LX and the EX for the desired routine (on the assumption that there is a conven-

tion as to the EX that the service provider uses in creating the entry-table entry for a given PC routine).

8. Once the entry table has been created and the PC numbers are available globally, the entry table can be connected. Since a system LX is used for connection, the entry table will be connected to all address spaces.

```
        LA    R0,1              load and save
        ST    R0,TKCOUNT          number of ETs to be connected
        ETCON TKLIST=TKL,       connect entry table                 +
              LXLIST=LXL,                                           +
              MF=(E,ETCONLST)
```

   (Note that the count, entry-table token and LX values are fullwords.)

   Now the service provider's services in the form of PC-cp routines are available to any address space, present and future, so long as the service provider is active and doesn't disconnect the entry table.

9. Programs in other address spaces must find or construct the PC numbers for the routines they wish to call. For a subsystem named HRLM, the code below shows a technique for finding the LX in an area appended to the subsystem's SSVT, creating a PC number for the routine having EX=4, and issuing the PC call (any parameter registers are assumed to have been set):

```
        L     R14,CVTPTR        point to CVT
        USING CVT,R14
        L     R14,CVTJESCT      point to JESCT
        USING JESCT,R14
        LA    R14,JESSSCT-(SSCTSCTA-SSCT) point to address of first   +
                                  SSCT, less offset of SSCTSCTA
        USING SSCT,R14
GETSSCT@ ICM  R14,15,SSCTSCTA   point to next SSCT
        BZ    NOSSCT            if none, skip
        CLC   SSCTSNAM,=C'HRLM' right subsystem name?
        BNE   GETSSCT@          if not, try next
        ICM   R14,15,SSCTSSVT   point to SSVT if any
        BZ    NOSSVT            if none, skip
        USING SSVT,R14
        L     R14,HSSVTLX2      load system LX
        DROP  R14
        LA    R14,4(,R14)       add EX=4
        PC    0(R14)            invoke PC routine
        ...
* No SSCVT for the subsystem was found.
NOSSCT  DS    0H
        ...
* The subsystem hadn't created an SSVT.
NOSSVT  DS    0H
        ...
```

   It should be mentioned that there are circumstances in which a PC instruction cannot be validly issued to obtain services. One occurs when the entry table of the service provider has been disconnected from the address space's linkage table by operating system clean up at the end of a job-step task (or at the end of memory). If an installation intends to provide PC routines that are to be invoked after a job-step task terminates, care must be taken to use only those services available through system-LX connections. So, for example, if an installation's SMF exit IEFACTRT were to invoke an installation-written PC routine, the connection should be through a system LX, because that connection persists across job steps (an entry table connected with a non-system LX

would have been automatically disconnected before the exit got control). Another situation where PC instructions cannot be issued is execution in secondary or home ASC mode.

10. The PC routine writes an installation-defined SMF record for billing purposes; this is not shown. The standard workings of PC routines, however, will be laid out in section 5.0, “PC Routines” on page 34.

11. The service provider, when there is no longer any need for its cross-memory services or when it does not wish to offer them any longer, removes the services by disconnecting and destroying the entry table.

```
* Token is that returned by ETCRE.
         ETDES TOKEN=TKVALUE,     destroy entry table                 +
               PURGE=YES,           across all linkage tables         +
               MF=(E,ETDESLST)
```

    The PURGE parameter is required because the linkage tables of all address spaces in the system are connected to the entry table. If any address spaces were in fact executing PC routines described by the destroyed table, the effect on callers would be, as they say, unpredictable, but an abend might result, especially if the routines were PC-ss. Inasmuch as the operating system does not provide a method for ensuring that there are no users of the service provider's cross-memory services when it is about to withdraw them, it is up to the service provider and any users to coordinate themselves so as to avoid problems from the withdrawal of services.

12. A system LX cannot be freed, but it is available for reuse, either by the service provider address space or, once its job-step task terminates and the LX is “dormant,” by a program in another address space. If the service provider intends to offer cross-memory services again—perhaps after restart—it should save the LX value where it can later be found for reuse, for example, in common storage or in a named token.

4.4 The steps for generally connected PC-ss routines

If the service provider wishes to provide generally available routines to run with its own address space as primary rather than with the caller's, many of the steps are similar to those for the PC-cp routines.

1. Since there are to be space-switch routines, the service provider address space must make itself non-swappable before any of those routines might be called:

```
* Key 0 and supervisor state are assumed for this macro.
         SYSEVENT TRANSWAP,                                           +
               ENTRY=BRANCH
```

    (Alternatively, the service provider's job-step program could be marked in the Program Properties Table with the non-swappable attribute.)

2. A system LX is reserved as shown in point 3 on page 22.

3. Since the PC-ss routines may be called from any address space and since they run in the service provider address space, the latter may need primary and secondary ASN authorization to any address space. This requirement applies in the connection of an entry table with at least one PC-ss routine and also at least one PC-ss *or* -cp routine for which the secondary address space is to be the caller's primary (that is, SASN=OLD in the corresponding ETDEF).[16] Since in every

authorization table the second entry, for AX=1, permits PT-ss, SSAR-ss and PR-ss (refer to Figure 2 on page 14), the service provider address space sets itself to AX=1 before creating the entry table.

```
LA    R2,1              load value of 1
AXSET AX=(R2)
```

4. The modules constituting the PC routines must be available. PC-ss routines may be in common storage, but unlike PC-cp routines, they can reside in the private area of the service provider address space. Thus, such modules might be loaded with ordinary forms of the LOAD macro (but recall the warning in point 4 on page 23 about unintended deletion), or they might be internal routines such as CSECTs in the service provider load module.

5. An entry table description list is prepared, as in the example on page 25. Both PC-ss and PC-cp routines can be described in a single list. This defines an entry for a PC-ss routine in the description list:

```
ETDEF TYPE=SET,          PC-ss routine                      +
      ETEADR=(R3),         at this entry element            +
      ROUTINE=(R2),        entry point                      +
      AKM=(0:15),          all values                       +
      EKM=(0:15),          all values                       +
      PKM=OR,              caller's PKM ORed with this EKM  +
      SSWITCH=YES,         PC-ss                            +
      SASN=OLD,            SASN is caller's PASN (default)  +
      STATE=SUPERVISOR     supervisor state
```

6. The entry table is created via ETCRE, as in point 6 on page 26.

7. A PC number table is created or, alternatively, the LX is made available in commonly addressable storage, as shown in point 3 on page 22.

8. The entry table is connected as in point 8 on page 27.

   When a connected entry table contains space-switch entries, there are important effects of the ETCON operation in addition to entry table connection:

   - Any subsequent job steps in the address space are prevented from using the LXRES or ETCRE services or the AXRES service to be described later.
   - The address space is terminated after the last job step (even if the address space is an initiator).
   - Following memory termination, the ASID is marked unusable until the next IPL. This may pose an operational problem if many ASIDs are consumed in testing a service provider and the ASID limit (the sum of MAXUSER, RSVSTRT and RSVNONR) is reached (consult *MVS/ESA Initialization and Tuning Reference*).

   These restrictions are imposed to preserve system integrity (see section 6.4, “Conserving resources” on page 43, for more details).

9. Programs in other address spaces call PC routines as shown in point 9 on page 27.

---

[16] An abend 052-516 will result from an attempt to connect with a system LX an entry table with such entries if the owning address space lacks primary and secondary authority.

10. A PC-ss routine will run in the service provider address space, and its function might include moving data to or from the caller address space. There is an example of this in point 2 on page 36.

11. The service provider eventually disconnects and destroys the entry table as in point 11 on page 28.

12. The service provider should resume its default AX. Resetting the AX may not be done before the entry table owned by the address space has been disconnected, since that entry table describes a PC-ss routine.

```
SLR   R2,R2              load 0, default value
AXSET AX=(R2)
```

13. The service provider makes its address space swappable:

```
* Key 0, supervisor state is assumed for this macro.
        SYSEVENT OKSWAP,                                        +
              ENTRY=BRANCH
```

4.5 The steps for specifically connected PC routines

It may be the case that a service provider wants to make certain of its cross-memory services available only in specific address spaces. An installation accounting system, for example, might have PC routines callable from every address space to record billing data, but only end-of-day jobs are supposed to read accumulated data as input to billing programs. Generally connected PC routines might of course do their own checking and simply terminate if they determine the caller ought not to have invoked them. The cross-memory facility, however, can make PC routines available in connected address spaces only.

1. The service provider, if it intends to provide PC-ss routines, must make itself non-swappable.

2. Specific connection of entry tables is not done in the service provider address space but rather in individual address spaces. There is a difficulty in that connection is an MVS cross-memory service requiring supervisor state or PKM 0-7 in the program invoking the service. One solution to this problem is a generally connected PC routine that would check the legitimacy of a connection request and would then carry out connection on the caller's behalf. Thus, the service provider would provide two kinds of PC routines: Generally connected routines for checking, connection and eventual disconnection, and specifically connected routines.

   (Alternatively, the service provider could schedule SRB routines to its dependent address spaces to perform connection and disconnection, or there might be SVCs to carry out these tasks. Generally connected PC routines, however, seem the most natural way of making other PC routines connected in particular address spaces.)

   The service provider gets a system LX as shown in point 3 on page 22. As an additional step, the service provider gets an non-system LX (the default) and saves it in common storage:

```
LA    R1,1               load and save 1
ST    R1,LXCOUNT2          as number of LXs needed
LXRES LXLIST=LXL,        get a non-system LX into LXVALUE     +
      SYSTEM=NO,           make default explicit              +
      MF=(E,LXRESLST)
L     R1,SAVESSVT        point to SSVT
USING SSVT,R1
MODESET EXTKEY=ZERO
MVC   HSSVTLL2,LXVALUE2  save returned token
DROP  R1
```

```
MODESET KEYADDR=SAVEKEY, back to usual key                    +
      WORKREG=1
```

3. The modules are loaded into common or private storage, as appropriate.

4. An entry table description list is prepared for the generally available routines, as in the example on page 25, and another description list is prepared for the specifically available routines, following the same procedure.

5. Two entry tables are created, in separate ETCRE operations. Each returned token is saved, that for specifically available PC routines being saved in common storage:

```
L     R1,SAVESSVT        point to SSVT
USING SSVT,R1
MODESET EXTKEY=ZERO
ST    R0,HSSVTET2        save token from ETCRE
DROP  R1
MODESET KEYADDR=SAVEKEY, back to usual key                    +
      WORKREG=1
```

6. If the service provider is going to offer PC-ss routines with the caller's primary address space as secondary (SASN=OLD), the service provider address space needs primary and secondary ASN authorization for those address spaces that will be connected specifically (the present example assumes that none of the generally available PC routines are PC-ss). The service provider address space could set its AX to 1, but since primary and secondary ASN authorization is needed only for some address spaces, the service provider elects to use a different AX value. To begin, an AX offset in every address space's authorization table is reserved:

```
MVC   AXCOUNT,=H'1'      1 AX needed
AXRES AXLIST=AXL         AX value is returned in AXVALUE
```

   (Note that the count and the returned value are halfwords.)

   The service provider's job-step task owns the reserved AX offset, whether reservation was done under that task or under another dispatchable unit (TCB or SRB) in the address space.

   The service provider copies the returned AX value to common storage, since local address spaces will need to know that value:

```
L     R1,SAVESSVT        point to SSVT
USING SSVT,R1
MODESET EXTKEY=ZERO
MVC   HSSVTAX2,AXVALUE   save returned token
DROP  R1
MODESET KEYADDR=SAVEKEY, back to usual key                    +
      WORKREG=1
```

   (Alternatively, the AX value might be part of the latent parameter of the PC routine that will perform connection.)

7. The service provider sets the AX of its own address space to the reserved value, to prepare for specific connection of the entry table for space-switch routines:

```
AXSET AX=AXVALUE
```

8. The service provider connects the table for generally available PC routines, using the system LX, as in point 8 on page 27.

9. An address space wishing to have the specifically available services calls a generally connected PC routine to connect those services. That routine, after making sure that the caller is requesting those services legitimately, starts by granting both primary and secondary ASN authorization to the service provider address space (otherwise there will be an abend in trying to connect the

second entry table). The authorization is made in terms of the AX value saved in common storage:

```
* For convenience, the PC routine uses the same DSECT to map its
* private-area storage as does the service provider
        L     R2,SAVESSVT        point to HRLM's SSVT
        USING SSVT,R2
        ATSET AX=HSSVTAX2,       AX value from AXRES                 +
              PT=YES,            grant primary ASN authorization     +
              SSAR=YES           grant secondary ASN authorization
        DROP  R2
```

(An alternative for finding the service provider address space's AX is the AXEXT service. This requires that the user of AXEXT first have determined what the ASID of the service provider is; in other words, the service provider must make that piece of information known or knowable.)

The routine proceeds by connecting the current address space's linkage table to the entry table for specifically available PC routines:

```
        LA    R0,1               load 1
        ST    R0,TKCOUNT         number of ETs for connection
        ST    R0,LXCOUNT         number of LXs to be connected
        L     R1,SAVESSVT
        USING SSVT,R1
        MVC   TKVALUE,HSSVTET2   copy ET for second entry table
        MVC   LXVALUE,HSSVTLL2   copy value of local LX
        DROP  R1
        ETCON TKLIST=TKL,        connect entry table                 +
              LXLIST=LXL,                                            +
              MF=(E,ETCONLST)
```

The specifically available PC routines may now be invoked by callers in this address space.

As with generally connected tables, there are consequences of specifically connecting a table that contains PC-ss entries:

- The ASID of the owning address space is not reusable until all connected address spaces have gone through job-step termination.

  Note: In the case of a TSO address space, it must have been terminated by LOGOFF or CANCEL. LOGON not preceded by LOGOFF (that is, reusing the address space) will not suffice.

- The LX reserved to connect the table is similarly not reusable.

> These restrictions are imposed to preserve system integrity (see sections 6.4, “Conserving
> resources” on page 43 and 6.5, “System integrity in resource conservation” on page 44, for more details).

10. Another generally connected PC routine should eventually be called by the local address space to disconnect the specifically connected entry table from the address space's linkage table (the table for generally connected routines cannot be disconnected by the local address space, as that disconnection requires destruction of the table and consequently is the responsibility of the service provider) and by removing primary and secondary ASN authorization:

```
LA    R0,1              load 1
ST    R0,TKCOUNT        number of ETs for disconnection
L     R2,SAVESSVT       point to HRLM's SSVT
USING SSVT,R2
MVC   TKVALUE,HSSVTET2  copy ET for second entry table
ETDIS TKLIST=TKL        disconnect entry table
ATSET AX=HSSVTAX2,      AX value from AXRES                 +
      PT=NO,            remove primary ASN authorization    +
      SSAR=NO           remove secondary ASN authorization
DROP  R2
```

> (With disconnection, too, there can be system-integrity concerns. Again, see section 6.5, “System
> integrity in resource conservation” on page 44, for more information.)

11. When all local address spaces have disconnected the specifically available table from their linkage tables, or when the service provider no longer wishes to offer its cross-memory services, the service provider destroys both its entry tables, using the appropriate tokens, as in point 11 on page 28. The ETDES for the specifically connected table does not require PURGE=YES if all connections have been broken.

12. The service provider address space assumes the default AX value, frees the reserved AX offset and frees the non-system LX (the system LX cannot be freed),

```
SLR   R2,R2
AXSET AX=(R2)           back to default
MVC   AXCOUNT,=H'1'     1 AX
L     R2,SAVESSVT       point to HRLM's SSVT
USING SSVT,R2
MVC   AXVALUE,HSSVTAX2  copy AX from SSVT extension
MODESET EXTKEY=ZERO
XC    HSSVTAX2,HSSVTAX2 clear AX value
MODESET KEYADDR=SAVEKEY,                                    +
      WORKREG=1
AXFRE AXLIST=AXL        free AX value
LA    R0,1
ST    R0,LXCOUNT2       indicate number of LXs
MVC   LXVALUE2,HSSVTLL2 copy non-system LX value
MODESET EXTKEY=ZERO
XC    HSSVTLL2,HSSVTLL2 clear non-system LX value
MODESET KEYADDR=SAVEKEY,                                    +
      WORKREG=1
DROP  R2
LXFRE LXLIST=LXL2,      free non-system AX value            +
      MF=(E,LXFRELST)
```

Note: If it were the case that not all address spaces had disconnected their linkage tables from the entry table connected with the non-system LX, the FORCE=YES parameter on LXFRE would be needed to free that LX.

13. The service provider address space makes itself swappable if it made itself non-swappable.

14. If the service provider wishes to repeat this entire process, it can do so, reusing the saved system LX.

5.0 PC Routines

There are some things a stacking-PC routine must or should do as part of its prolog and epilog. For example, the routine must respect register conventions and take into account possible changes from the state of the caller to the state in which it gets control. This section lays out those things that writers of PC routines should know.

5.1 Entry to the PC routine

1. Register conventions are different from the usual MVS conventions and also different from SVC conventions. When a stacking-PC routine receives control, registers have these contents:

   R0-R3, R5-R15 and ***AR0-AR15***
   Are unchanged from their state when the caller issued the PC instruction. R0, R1 and R15 often serve as parameter carriers; if the caller was in AR ASC mode, the corresponding ARs may contain ALETs to qualify addresses in those registers. R13 typically points to the caller's save area and should not be used to save status.

   R4
   Points to the latent parameter values from the entry-table entry describing the PC routine.

   Note that there is no register containing the PC routine's entry point. For addressability, something like a `BASR Rx,0` must be done by the routine.

2. There may have been state changes as the result of the PC instruction:

   - In the case of a PC-ss routine, the PASN of the routine is that of the owning address space; the SASN is the PASN of the caller if the routine was defined with SASN=OLD (the default), or is the new PASN if SASN=NEW applied. In the case of a PC-cp routine, the PASN is unchanged, and the SASN is the PASN.
   - The routine may have received control in supervisor state when its caller was in problem state, or the converse.
   - The routine's PKM may have increased by the ORing of the caller's PKM with the EKM from the entry table for the routine, or the PKM may have been replaced by that EKM.
   - The PSW key may have changed.
   - The ASC mode may have changed from primary to AR, or the converse.
   - AX is that of the current primary and may have changed if there was a space switch.
   - The EAX may have changed.
   - An ARR recovery environment may be in force.

   The following will *not* have changed:

   - Enablement or disablement.
   - Locks held.
   - Job-step (APF) authorization, or lack thereof.
   - The home address space.

- The dispatchable unit (TCB or SRB), which is part of the home address space. Note that execution time in the PC routine is charged to the dispatchable unit.

3. Status does not need to be saved, since it will have been put into the linkage stack: The caller's general-purpose and access register values as well as PSW are available. To get the caller's R1 value and PSW, for example:

```
EREG  R1,R1           get R1 (and AR1) from linkage stack
LA    R3,1            load 1 to designate PSW
ESTA  R2,R3           get PSW from linkage stack into R2-R3
```

4. The PC routine should have an ARR or more probably would establish a recovery environment by ESTAE or SETFRR, especially if the routine is going to do work that has integrity or consistency requirements. Note, however, that there may be a restriction on issuing SVCs to instate a recovery environment (see point 1 in section 5.2, “PC Mainline”).

5. The latent parameter is two contiguous fullwords to which general-purpose register 4 points at entry. If the PC routine was defined without the first or second latent parameter value, the corresponding fullword will contain zero (since not specifying a value is equivalent to specifying a value of zero).

   When the second PC routine defined on page 25 is entered, R4 would point to copies of the first value (defaulted to zero) and the second value, namely,

```
DC    F'0'            no value given
DC    X'12345678'     second value given
```

   To get the second value, the PC routine would do:

```
ICM   Rx,15,4(R4)     load second latent parameter fullword
BZ    wherever        skip ahead if value is zero
...                   do whatever the parameter is for
```

6. If the PC routine has a work area, the convention (useful in debugging) is that R13 points to it and that area's second word contains the string F1SA. If the routine has no save area, the convention is that R13 contains zero.

   Note: If storage is obtained and is task-related, it is associated with the address space that is primary when it is obtained. Consequently, care must be taken to free the storage in both normal and abnormal execution paths, lest PC-ss routines exhaust private-area storage in the service provider address space. Furthermore, care must be taken that the owning task not terminate prematurely, lest the storage used by the PC-ss routine but owned by the task be freed whilst a PC-ss routine is running.

5.2 PC Mainline

1. There are some restrictions on what a PC routine (or any other cross-memory program) can do:

   - Almost all SVCs may not be invoked if the calling program is in cross-memory mode; there are other MVS services with this restriction, too. Many services are, however, available through PC calls (e.g., STORAGE), and branch entry is often available. Macro descriptions usually tell in what states MVS services may be invoked.

   - If the routine is going to make secondary an address space that isn't home or that wasn't the primary before the PC call, the intended address space must not be swapped out (otherwise, an ASID-translation exception and 0D5 abend follow). To prevent swapping out of the address space, it should be the home address space, or the address space's local lock should be held in the form of the CML lock.

Page faulting, in contrast to access to storage in a swapped-out address space, is permitted.

- The routine should not use MODESET to enter problem state, as the MODESET SVC will set PKM to match the PSW key. If the PC can issue SVCs, SYNCHX may be employed to give control to code in problem state whilst preserving the PC routine's PKM.

2. The PC routine proceeds to its substantive work. In general, that work can be whatever an SVC routine may do, if the PC routine runs in supervisor state or has PKM 0-7. A generally available PC-cp routine might use ATSET, ETCON and ETDIS to make specifically available PC routines usable by the routine's caller, since the effect of those services persists after return from the routine. The routine could access, update and copy data between the primary address space and any address space to which it had secondary access as a result of the caller's PC or of SSAR in the routine (remember that a PC routine, like any other program, is subject to primary and secondary authorization). The routine could call other PC routines.

   A program might call a PC-ss routine that gets control with the caller address space as the secondary, in order to copy data to or from the caller address space. For example, a caller running in key 8 may have placed a value into a field and expects that field and the following field to be updated by the PC routine:

```
         LA    R1,PARMLIST        point to parm list
         PC    0(R14)             call PC-ss routine
         ...
* data areas
PARMLIST DS    0F
PARM1@   DC    A(PARM1)
PARM2@   DC    A(X'80000000'+PARM2) high-order bit indicates end
PARM1    DC    CL8'JamesA'        input and output parameter 1
PARM2    DC    CL8                output parameter 2
```

   The PC routine loads the parameter from the first field, PARM1, and updates that field and the following field. By way of illustration, the loading and first update are performed in secondary ASC mode, and the second update is done in primary ASC mode:

```
* The "normal" PSW key is this example is 4.
         LA    R1,1               value for obtaining stacked PSW
         ESTA  R0,R1              get stacked PSW
         SRL   R0,16              get caller's PSW key into bits 24-27
         LR    R9,R0              copy caller's PSW key information
         EREG  R1,R1              get R1 from linkage stack
         IAC   R5                 save current addressing mode
         SAC   X'100'             go to secondary ASC mode
         SPKA  0(R9)              to caller's key
         L     R2,0(,R1)          point to first field
         LM    R6,R7,0(R2)        load first field from caller
         MVC   0(8,R2),=CL8'Frodo'  copy some stuff back to that field
         L     R2,4(,R1)          point to second field
         SAC   0(R5)              resume previous addressing mode
         LA    R5,8               load true length
         MVCS  0(R5,R2),=CL8'Bilbo',R9  copy string to second field
         SAC   X'100'             back to secondary ASC mode
         TM    4(R1),X'80'        end of list?
         BZ    wherever           no, do further list processing
         SAC   0                  yes, back to primary ASC mode, with  +
                                    explicit operand
         SPKA  X'40'              back to "normal" PSW key
         STM   R6,R7,wherever     save stuff
```

   There are several things that may not be obvious in this example, which is more complex than it may appear:

- The PC routine must be resident in non-fetch-protected common storage since the code fragment starts in PSW key 4, goes to secondary ASC mode, and then in PSW key 8 fetches data (from the secondary address space, of course) that include both caller-specified data (e.g., the first field) and inline data (the character constants in the PC routine module). (Instruction fetch in secondary ASC mode is of course from the primary address space.)
- The routine is allowed to use the secondary-space access key (i.e., key 8) in R9 in the MVCS instruction because the routine is running in supervisor state (the routine's PKM, of course, allows the access key, but PKM checking is bypassed in supervisor state).
- The routine has access to its caller's *storage* in the MVCS instruction because the secondary-space access key matches that of the storage. If, on the other hand, the PKM permitted a certain secondary-space key but that key did not match the storage key, an 0C4 would occur. An example: If the routine had chosen key 1 as its secondary-space key, that key would pass cross-memory authorization checking—the routine's PKM, ORed from the EKM of the entry table entry as prepared in the example in point 5 on page 29, allows that much—but the secondary-space key would not pass storage protection checking.
- The routine assumes secondary ASC mode because it wishes to be able to follow a chain of information (the two pointers and the end-list bit in the parameter list): Although the routine could have used MVCP to copy the chain or pieces of the chain into its address space and then employed MVCP and MVCS to copy and update fields, it is less awkward to assume secondary ASC mode and so to follow the chain, load a field and update the field in direct operations.
- Primary or secondary ASC mode must be in force at the beginning of the code fragment, since MVCS does not work in AR or home ASC mode.

The same effect is achieved more intelligibly with access registers:

```
LA    R1,1               value for obtaining stacked PSW
ESTA  R0,R1              get stacked PSW
SRL   R0,16              get caller's PSW key into bits 24-27
LR    R9,R0              copy caller's PSW key information
EREG  R1,R1              get R1 (and AR1) from linkage stack
LAM   AR0,AR15,=16F'0'   ensure ARs are zero
LA    R0,1               load ALET for secondary addr space
SAR   R1,R0              load ALET into AR1
SAR   R2,R0              load ALET into AR2
SPKA  0(R9)              to caller's key
SAC   X'200'             go to AR ASC mode
L     R2,0(,R1)          point to first field
LM    R6,R7,0(R2)        load first field from caller
MVC   0(8,R2),=CL8'Frodo'  copy some stuff back to that field
L     R2,4(,R1)          point to second field
MVC   0(8,R2),=CL8'Bilbo'  copy string to second field
TM    4(R1),X'80'        end of list?
BZ    wherever           no, do further list processing
SAC   0                  yes, back to primary ASC mode, with  +
                           explicit operand
SPKA  X'40'              back to "normal" PSW key
STM   R6,R7,wherever     save stuff
```

5.3 Exit from the PC routine

There are several steps in the PC routine's exit process:

1. Task-related storage obtained by the routine should be freed, instead of relying on end-of-task clean up. Freeing is especially important if the storage was obtained in the service provider address space, since the storage is owned by the latter and not by the caller's address space.
2. An explicitly established recovery environment should be deleted.
3. A return code may be put in R15, as may other information into R0 and R1 (and AR15-AR1).
4. The routine must be in primary or AR ASC mode. PASN, SASN, PSW key and problem/supervisor state are unimportant. Return to the caller is effected by a PR instruction.

   ```
   PR    ,                  return and restore status
   ```

   The instruction has the effect of restoring the caller's state:

   - R2-R14 and AR2-AR14 are restored from the linkage stack. (R15-R1 and AR15-AR1 are not restored.)
   - Return address, AMODE and problem/supervisor state are set in the PSW from the stack.
   - PASN, SASN, PKM and EAX are restored from the stack.
   - AX will be that of the new PASN.

5.4 Recovery and retry

An ESTAE-type recovery environment for a PC routine may be provided by the service provider's definition of an Associated Recovery Routines (ARR). The PC routine may itself instate a similar environment by ESTAE/ESTAEX, or it may instate the more powerful environment of a Functional Recovery Routine (FRR). The reader is assumed familiar with recovery and retry in general, and so only salient features are discussed.

5.4.1 Associated recovery routines (ARRs)

An associated recovery routine (ARR) is very much like a recovery routine provided by ESTAEX. If an ARR is specified for a PC routine, an ARR environment is in force upon entry to the PC routine. (In this respect, the ARR environment is like that of ESTAI: Each is effectively established[17] when the subject routine gets control.) Consequently, the PC routine does not need to use ESTAE if the ARR environment will suffice. An ARR environment has these characteristics:

- The ARR module must be addressable at the point of error. This means that it must reside in common storage if a PC-cp routine is involved; for a PC-ss routine, the module may be in the service provider address space's private storage.
- The ARR routine will not get control after an error if the caller of the PC routine is executing in SRB mode, is locked or has an FRR in force. The ARR will get control if the caller is disabled.

[17] The statement in *MVS/ESA Application Development Guide: Authorized Assembler Language Programs, SP4*, GC28-1645-1, page 16-16, about establishment is misleading as that concept relates to supervisor/problem state and PSW key at entry to an ARR. In the context of ARRs, recovery is effectively established by *invocation* of a PC routine for which an ARR has been *defined*.

- If a task-mode PC routine establishes an ESTAE-type environment, the ARR will get control after the ESTAE-type recovery routine if the latter elects percolation. The ARR will get control before any ESTAE-type recovery routine of its caller (including an ARR).
- If the PC routine establishes an FRR environment,[18] the ARR will get control if the FRR elects percolation (and if the caller was not SRB-mode or locked and had no FRR).
- The ARR will get control in PSW key zero if the PC routine is defined to get control in system key (0-7), and will get control in supervisor state if the PC routine is defined to get control in supervisor state. The ARR will get control in 31-bit addressing mode, enabled and unlocked; PASN and SASN and ASC mode will be as at entry to the PC routine.
- The linkage stack will be that at the point of error. This could be different from that at entry to the PC routine.
- SDWAPARM (if an SDWA is available) will point to a copy of the modifiable area of the linkage-stack entry created when the PC routine was invoked. (Hence this area allows the PC routine to communicate with its ARR.)
- The ARR environment will not permit retry in disabled or locked mode. Retry, if specified, will be with PASN, SASN, ASC mode and linkage stack as at entry to the PC routine.
- If the entry table defining a PC routine with an ARR is deleted and an error occurs in the routine (which must of course have been running before deletion), the ARR will not get control, because deletion will have removed information needed for recovery.

5.4.2 ESTAE-type recovery routines

A PC routine may establish an ESTAE-type recovery environment if the execution environment permits (for example, no locks may be held in trying to establish that environment). ESTAEX is to be preferred, as its valid environments include AR ASC mode, PASN¬=HASN and SASN¬=HASN and, further, because recovery and retry will be with PASN, SASN and ASC mode as at the point of establishing the environment. The recovery and retry environments—PASN, SASN and so forth—are like those for ARRs.

5.4.3 Functional recovery routines (FRRs)

An FRR environment allows both recovery and retry in many execution modes, including ones where the caller of the PC routine is disabled, locked or in FRR mode. The rules for the linkage stack at recovery and retry are the same as for ARRs and ESTAE-type routines. (Note that there is no intended interface to determine whether FRR mode is in force.) Because of this capability, it may be better for the PC routine to establish an FRR rather than for the service provider to define an ARR for that PC routine. On the other hand, an FRR environment precludes using some system services, for example, those requiring an SVC and those prohibiting callers with FRRs.

---

[18] The statement in *MVS/ESA Application Development Guide: Extended Addressability, SP4*, GC28-1652-2, page 3-37, that a "PC routine must have no FRRs" is mistaken. Probably what was meant is that if the caller of a PC routine has an FRR, the PC routine's ARR will be ignored.

6.0 Miscellaneous Topics

6.1 PC routines versus SVC routines

A stacking-PC routine will generally offer greater flexibility than an SVC routine. To show the advantages of one over the other, PC and SVC routines are contrasted here.

	PC routine	***SVC routine***
The first set has to do with making the routines available.		
1.	A rather large number (up to 2^{20}) of PC routines may be defined.	The number of SVC routines is 256 plus 256 that can be mediated through the SVC router.
2.	The module can reside in common storage (PC-cp) or private storage (PC-ss).	The module resides in common storage.
3.	A PC routine is effectively available from the time when its entry table is connected to the address space that is the caller's primary, and the connection lasts from that time until the service provider disconnects the table or terminates.	An SVC routine is defined from early in IPL or from the time that an authorized program uses SVCUPDTE to change the SVC's definition.
The next set involves invoking the routines.		
4.	Insofar as invocation goes, a caller may be locked, disabled, in task-, SRB- or other mode (e.g., an interrupt handler exit) and in cross-memory mode and may have an FRR. (The PC routine must of course be able to execute in such states, or it must return to the caller with return/reason codes if it cannot.)	Except for certain calls (SVC 3 and SVC 13), a caller must be unlocked, enabled, in task mode and not in cross-memory mode and may not have an FRR.
5.	A caller may be in primary or AR ASC mode.	A caller must be in primary ASC mode unless the SVC supports AR ASC mode.
6.	A caller may be required to be in supervisor state and/or have a certain PKM.	A caller may be required to be in supervisor state, PSW key 0-7 and/or APF-authorized (job-step authorization).
The next set concerns how the routines get control.		
7.	Entry and exit are effected through hardware.	Entry and exit are mediated through software.

8.	The caller's state is saved in the linkage stack.	The caller's state is saved in various places (e.g., PRB and SVRB), depending on the type of SVC.
9.	No locks are obtained at entry, but a PC routine can obtain a lock if it can get to supervisor state and PSW key 0.	An SVC routine can be defined to hold the local or CMS lock when it is given control.
10.	An ARR routine can be defined to be in force at entry. (ESTAE or FRR recovery can be established after entry.)	At entry, no recovery environment is defined, but an SVC routine can establish such.
11.	Either supervisor state or problem state can apply at entry.	Supervisor state applies at entry.
12.	A certain PSW key can apply at entry, or the caller's PSW key can continue to apply.	PSW key 0 applies at entry.
13.	A certain EAX can apply at entry, or the caller's EAX can apply.	The caller's EAX applies.
14.	The PKM may be defined to replace the caller's or to be ORed with the caller's.	The caller's PKM applies.
15.	Either primary or AR ASC can apply at entry.	Primary applies at entry.
16.	Not applicable.	An SVC can be defined as preemptible or non-preemptible by I/O interruptions.
17.	The PASN can be that of the caller (PC-cp) or that of the service provider (PC-ss).	The PASN is that of the caller. An SVC routine may be able to establish a different PASN.
18.	One or two latent parameters may be defined to be available at entry.	Not applicable.
19.	Only R4 changes from the caller's general-purpose and access register values.	Certain general-purpose registers contain pre-defined values (e.g., the CVT address in R3).

6.2 PC-ss routines versus SRB routines

It may be thought that a PC-ss routine is a general replacement for an SRB routine. The two are in fact very different creatures. At entry to a PC routine the primary and secondary address spaces are typically different, whilst they are the same at entry to an SRB routine. The PC routine cannot be validly called unless the server address space is swapped in, whilst dispatching an SRB ensures that the target address space is swapped in. A PC routine executes under its caller's dispatchable unit, but an SRB is a new and distinct dispatchable unit. Some resources in a PC routine are attributed to the primary (e.g., private storage) and some to the home (e.g., CPU time); in an SRB routine, primary and home are the same.

Given these characteristics, a PC-ss routine and an SRB routine will tend to be employed for different ends. A PC routine would serve to access data in the server and in the caller and might copy data back and forth; this dual access is the routine's strong point vis-à-vis an SRB, for an SRB routine

would not ordinarily have access to the caller address space (strictly speaking, to the scheduling address space). The SRB's strong point is that it is a separate dispatchable unit: It can cause the target to be swapped in and will run independently of the scheduler. Since an SRB is independent, many things follow: Interference with the scheduler is minimized (for example, a page fault or anything that temporarily halts the SRB will have no effect on the scheduler), but coordination with the scheduler is more difficult, if coordination is needed; the SRB has ready access to the target's control blocks such as TCBs and RBs, since these are associated with the dispatched address space; resources are attributed to the SRB or to its address space; errors in the SRB routine ordinarily do not affect the SRB's address space, much less that of the scheduler.

6.3 EAX

As seen in point 2 on page 36, it may be convenient for a PC routine to enjoy access to the service provider address space via access registers. But since that address space must have first been made available through an access list, there could be integrity or security problems because all programs under a dispatchable-unit access list (DU-AL) or a primary-address-space access list (PASN-AL) enjoy the access the list indicates. Since adding and removing access-list entries by a PC routine each time it executes would degrade performance, a further condition for access in AR ASC mode can be enforced: Having a certain EAX.

For a PC-cp or PC-ss routine to use a non-zero EAX:

1. An AX is reserved by the service provider. This AX may be the same as or different from an AX reserved for giving primary and secondary ASN authorization in dependent address spaces (see section 3.5.3, “The steps in offering and withdrawing cross-memory services” on page 15).
2. The authorization table of the service provider address space is set to confer secondary ASN authorization.
3. The service provider address space makes itself non-swappable.
4. The service provider defines an entry table with the PC routine having the AX as an EAX, and the table is connected generally or specifically.
5. A program in a connected address space uses ALESERV to create a *private* entry for the service provider address space either in the PASN-AL or in the DU-AL of the TCB or SRB under which the PC routine is to run. (The program creating the access-list entry might or might not be the same as the PC routine having the EAX; if it is not, CHKEAX=NO would be specified on ALESERV.)
6. The PC routine having the EAX can use instructions in AR ASC mode for access to data in the service provider address space. Programs not having that EAX, however, will not enjoy access.

To undo the above, the steps are reversed, except that instead of ALESERV to delete the access-list entry, it suffices for the connected address space's job-step task to go through termination or, if a DU-AL was involved, for the corresponding dispatchable unit to go through termination.

6.4 Conserving resources

A service provider address space that offers cross-memory services may consume resources to the point that system services or its own services become unusable. There are several things to know:

- The number of system LXs is not especially large, and a service provider should reuse any reserved LX, since it is never free before the next IPL. This advice applies with especial force to service providers that are often taken down and up, as may happen in code development.
- The number of non-system LXs is rather larger but not unlimited, and they can become non-reusable in that temporarily or permanently, they will not be reassigned. If an address space owning a PC-ss entry table goes through termination of its XMRO task and if there are outstanding “latent binds” to the address space because of connections to the entry table, the LXs used for connection will not be reusable until all the connected address spaces also go through XMRO task termination. If a connected address space is long-running—for example, CONSOLE, JESx, CICS—non-system LXs may not be reused until the next IPL.
- ASIDs of address spaces owning PC-ss entry tables may not be reused whilst there are latent binds to the owning address space. If a table was connected by a system LX, the ASID is not reused until the next IPL. Neither will the ASID be reused if connection has been made by a long-running address space. Finally, the ASID will not be reused if an owning address space's entry table is connected by a non-system LX to an address space itself owning a table connected by a system LX; for a program in any address space could invoke a PC routine running in the latter, and that PC routine could invoke a PC routine running in the former.
- AXs, when used as EAXs, are subject to non-reuse for the same reasons as ASIDs: Latent binds must not cause integrity problems. Such AXs will not be reused by the operating system until the owning address space terminates.
- A PC-ss routine may obtain working storage from the private area of the service provider address space. That storage will belong to a task that is not likely to terminate before the service provider (otherwise, storage would disappear from underneath the PC routine, as it were). The PC routine must be careful to free the storage (even in error cases), lest private storage be exhausted and subsequent instances of the routine fail.

The reader will note that most of these potential problems are due to PC-ss routines: To ensure integrity of the system, a resource will not be reused until there is no possibility of a program continuing to employ the resource. Without non-reusability, a program might continue to use the resource when it represents an a service provider entirely different from what the program believes.

Several remedies may be employed to conserve these resources:

1. Reusing system LXs.
2. Minimizing the number of times an address space owning a PC-ss entry table connected by a system LX is brought down and started again.
3. Being careful about connecting PC-ss entry tables via non-system LXs to long-running address spaces. Disconnection should be ensured, for example, by having a resource manager that, upon termination of the XMRO task of the owning address space, schedules SRBs to ensure disconnection of connected address spaces.

4. Reusing a reserved AX as an EAX instead of reserving another AX for that purpose, until the reserving address space terminates.[19]

5. Freeing working storage in recovery or retry.

6.5 System integrity in resource conservation

Conserving resources may present problems in system integrity. One interesting case arises with selectively connected PC-ss entry tables. If the entry-table owner knows that a specific client address space is to be connected (for example, supporting certain RRS exits by connecting a PC-ss table of an RRS application to the RRS address space), the owner may schedule an SRB routine to perform connection. If, however, the task (or address space) owning the entry table terminates before the SRB routine has connected the table, it is possible for the LX and EX to be freed (if not connected to yet another client address space) by the system and then reused for some other, entirely unrelated table: When the SRB routine at last effects connection, the wrong table is connected! Worse, the owner of the new table might attempt to connect it to that same client and suffer an abend in trying to connect an already connected table. This would be an error that, from the second owner's point of view, came out of the blue, and that owner might choose to terminate.

Here is the outline of one solution employed in IBM code:

1. The entry-table owner has a RESMGR for end of job-step task and one for end of memory. It is guaranteed that the system will not clean up, and possibly reuse, cross-memory resources like LXs belonging to the task or address space until after the EOT or, in the case of address space failure, the EOM RESMGR has run.

2. The owner builds a parameter list in common storage for the connecting SRB routine and schedules the routine.

3. If the RESMGR is entered, it checks to see whether any connecting SRB routine is “in flight” and, if so, sees whether it has yet to indicate connection was effected. If connection has not been indicated, it is crucial that the RESMGR not complete (and thereafter cross-memory clean up occur) until it is certain the SRB routine can no longer run.

   The RESMGR builds a parameter list with its ASID and an ECB, schedules a different SRB routine to the client address space and waits on the ECB. (The RESMGR's actions after the wait are described below.) This latter SRB routine schedules an IRB onto a convenient task such as the RCT and terminates. The IRB issues PURGEDQ to ensure that the connecting SRB, if not dispatched for the first time, does not get control and that the SRB, if dispatched but suspended (e.g., for a page fault) or dispatched and executing, gets abended. The IRB routine posts the RESMGR's ECB and completes.

4. The connecting SRB routine sets a bit in its parameter list to indicate it is about to attempt connection. If connection succeeds, it sets a bit indicating connection was made. If it incurs an abend before setting the latter bit, it specifically does *not* retry connection. In fact, a PURGEDQ-related abend incurred in the midst of connection leaves the SRB, and the RESMGR, in ignorance as to whether the system succeeded in making the connection but had not yet returned to the SRB routine.

[19] If the EAX can still be in use when the owning job-step task terminates, the address space will terminate after the last job-step task.

> 5. The client address space might fail before the IRB had posted the ECB, so leaving the RESMGR waiting forever. This might not be much loss in the case of an EOT RESMGR, but waiting forever by an EOM RESMGR could adversely affect the whole system's operation. The necessary RMTR logic to post the ECB is left as an exercise for the reader.

> 6. The back side of this general problem is to disconnect and thus ensure that the LX is recycled after any connections were made or attempted (for the latter, see the case above of ETCON interrupted by PURGEDQ). Here the solution is relatively straightforward. *Before* destroying the entry table, the owner schedules an SRB to effect disconnection in the client. It is important that the table still exist during the attempt at disconnection, lest a race condition arise where the client connection was never made, the table is destroyed, the LX is recycled and used by some other application to connect its entry table to the same client: The first owner's SRB would then disconnect the second owner's table! But so long as the table exists, the LX will not be recycled. (If there was no connection, the disconnecting SRB routine will incur an abend, but it can recover, retry back to mainline and exit.)

6.6 The PT instruction

PT is fundamentally a branch instruction that may change PASN and SASN. (The PC instruction of course is also a branch instruction that may change PASN and SASN, in addition to other state changes.) PT can be used as a quick way to change primary (and secondary) address spaces if the issuer is in common storage and enjoys authorizing AX values in all potential primaries. If for example a routine were in common storage and wanted a quick switch to primary addressability amongst several address spaces (all of them swapped in), PT could be employed to effect the change of primary (and secondary) address spaces.

The following is an example. The reader should understand that control registers are not an intended programming interface; control register 4 is used here because the example is taken from a server prototype and because the dependency of the PT instruction on control register 4 is made clearer. If the caller address space had AX=1 (as does the server's address space), no change to control register 4 would have been needed.

```
*************************************************************************
*                                                                       *
* Routine is in common storage, AMODE(31), with AX=1, disabled.  AX=1 *
* guarantees the first PT will work, and disablement prevents CR4     *
* from changing due to an interrupt after CR4 has been loaded below.  *
* The routine is in supervisor state (to allow STCTL and LCTL).       *
*                                                                       *
*************************************************************************
         ICM   R3,3,TARGPASN      load PASN of target
         ICM   R3,12,=X'FFFF'     load maximum PKM for ANDing with CR3
         LA    R14,THERE          point to switch-to point
         O     R14,=X'80000000'   indicate 31-bit mode
         SAC   0                  ensure primary ASC mode
         PT    R3,R14             change primary addr space
*************************************************************************
*                                                                       *
* Executing in primary of caller.                                       *
*                                                                       *
*************************************************************************
THERE    DS    0H
         ...
         ICM   R3,3,SRVRPASN      load PASN of original addr space
         ICM   R3,12,=X'FFFF'     load maximum PKM for ANDing with CR3
```

```
         LA    R14,HERE           point to switch-back point
         O     R14,=X'80000000'   indicate 31-bit mode
         STCTL CR4,CR4,SAVECR4    save CR4
         MVC   SAVECR4(2),=X'0001' max AX value 1
         LCTL  CR4,CR4,SAVECR4    load new value
         SAC   0                  ensure primary ASC mode
         PT    R3,R14             change primary addr space
*********************************************************************
*                                                                   *
* Executing in primary of service address space.                    *
*                                                                   *
*********************************************************************
HERE     DS    0H
         ...
```

6.7 The CML lock

- Is the local lock of an address space other than home and is not the CMS lock.
- Is a suspend lock.
- May not be requested by a program holding the local lock.
- May be requested only for an address space that is primary or secondary; this effectively means the target must be swapped in and usually means the target is non-swappable. The CML lock of the *MASTER* address space may not be requested.

6.8 Performance considerations

The cost of setting up the cross-memory environment is not trivial, and PC and PR instructions are relatively expensive in comparison to instructions like BALR. On the other hand, PC routines will typically have a performance advantage over SVC and SRB routines that might accomplish similar purposes, because SVCs have front-end overhead in going through the SVC interrupt handler and back-end overhead in going through exit processing, and SRB routines have overhead in scheduling, dispatching and exit processing. PC routines do not go through an interrupt handler or the dispatcher when they receive control, neither do they go through exit processing like that of SVC or SRB routines. Thus, PC routines will tend to perform better than SVC and SRB routines for comparable functions.

The real-memory cost of non-swappable address spaces should be considered when designing PC routines or other programs that would access other address spaces. Parts of those memories are backed by real frames so long as non-swappability lasts.

For programs that can be trusted to use it safely, execution in supervisor state gives better performance because some hardware checking of PKM is avoided.

7.0 Debugging Cross-memory Programs

This section is a miscellany of hints, tips and observations, inasmuch as there isn't a systematic approach to debugging cross-memory services that is much different from any other systematic approach for debugging programs that are effectively extensions of the operating system.

- Debugging cross-memory programs is frequently difficult if there is a need to examine storage both in a dependent address space and in the service provider address space. (The matter is rather easier if only one address space is involved.) Dumps such as SYSMDUMP may suffice for the dependent address space, but these won't include storage in the service provider address space. For that purpose, a SYSMDUMP or SVC dump in the service provider address space will be required.
- Another difficulty that stems from dumping private-area storage is that most cross-memory tables are in PCAUTH's private area. They can be inspected by taking an SVC dump of PCAUTH. A few things to note:
  - Since hardware employs real addresses in support of the cross-memory facility, some addresses in PCAUTH's tables are real.
  - An entry-table entry will consist of 32 bytes if control register 0's ASF bit is one, and this is always true in MVS/SP3 and higher, since the stacking process that is part of the PC and BAKR instructions depends on the bit being one.
  - Entry tables are filled out to a multiple of four entries if necessary. Thus, there may be some invalid entries. Invalid entry table entries can be quickly spotted because the latent parameter pointer will be zero.
- A summary of the entry tables to which an address space is connected and of the address space's authorization table can be found in an SVC dump of the address space by using the IPCS subcommand SUMMARY FORMAT.
- The SLIP command can be a powerful debugging tool, especially because an SVC dump of more than one address space can be requested, and the specification of address spaces may be in terms of home, primary, secondary and current address spaces.
- In the case of an abend in a PC routine, registers and PSW will be available in dumps whether the routine was PC-cp or PC-ss. PASN and SASN should appear, too. The linkage stack may be displayed by the IPCS subcommand CBFormat 0. STR(LS).
- If the entry-point address in a PC routine's entry table entry was invalid and the ETCRE service did not catch this, an abend will result from the PC call. The abend might take the form of an 0C1 or, if there was no valid page at that address, an 0C4.
- The cross-memory status of a program (task or SRB) can be found in the XSB summary in the routine's dump. An address space's cross-memory status (type and number of resources, AX value) may be discovered in various ASCB and ASTE fields.
- Records from the system trace table may be quite useful, especially because there are records for each PC, PR (from a PC, but not from a BAKR, since BAKR itself is not traced), PT and SSAR. In a SYSMDUMP dump, however, not all trace records of interest may appear, since it may not be apparent to dump services that some records pertain to the address space being dumped. It may be necessary to dump the trace table in a different way, such as an SVC dump of other address spaces, to produce a more complete listing of its contents, and the table in the dump can be formatted by the IPCS subcommand VERBEXIT TRACE. (Note that in PC, PR and PT

records, the return address will be the return address *plus 1* if the issuer of the PC instruction was in problem state, because bit 31 in the return address actually denotes problem state.)

- The TEST command can be used in a limited way in TSO testing. Breakpoints cannot be placed in PC routines, though, if those routines are in non-key-8 storage (much less if they are in another address space), and breakpoints cannot be placed at PC instructions themselves. TEST will not stop at breakpoints if control is in supervisor state. If a PC routine abends, TEST will not restore the cross-memory environment, particularly if the PC routine was in cross-memory mode.

  The TESTAUTH command, on the other hand, allows testing of a PC routine itself, albeit in a very limited way. Breakpoints can be put into the routine if it has been loaded into key-8 storage. Control will stop at those points, possibly with a supervisor-state, system-key PSW. TESTAUTH may not, however, succeed in resuming execution correctly.

- A even more powerful debugging tool is to run MVS as a VM guest. Execution can be readily stopped by the CP command TRACE INSTR. Here it is helpful if the PC routine's module has been loaded into storage on a page boundary, because it is then easier to calculate offsets of instructions in the module. If the PC routine is not in common storage but rather in the service provider address space's private area, the TRACE command will require qualification by an ASTE or ASN. If the guest machine has several virtual CPs, the CP command CPU ALL should be used to issue the TRACE command.

8.0 References

K. E. Plambeck, “Concepts of Enterprise Systems Architecture/370,” *IBM Systems Journal*, vol. 28, no. 1, 1989. A lucid explanation of addressing and its history.

Enterprise Systems Architecture/390 Principles of Operation, SA22-7201. The bible for cross-memory.

C. E. Clark, “The facilities and evolution of MVS/ESA,” *IBM Systems Journal*, vol. 28, no. 1, 1989. A *tour d'horizon* of addressing from a software perspective.

MVS/ESA Application Development Guide: Authorized Assembler Language Programs, SP4, GC28-1645. Description of many authorized services.

MVS/ESA Application Development Guide: Extended Addressability, SP4, GC28-1652. A very helpful introduction to PC routines, the linkage stack, access lists, ALETs, EAXs and associated instructions.

MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs, SP4, volumes 1-4, GC28-1647 through -1650. Specification of many authorized services.

MVS/ESA Initialization and Tuning Reference, SP4, GC28-1635. Description of parameters used at IPL to limit the number of ASIDs (e.g., RSVNONR).

MVS/ESA Interactive Problem Control System (IPCS) Command Reference, SP4, GC28-1632.

James Antognini, “Software Subsystems in MVS,” SHARE 61 *Proceedings*, 1983. All about classic subsystems.

James Antognini, “Writing Authorized Programs with System Integrity,” SHARE 77 *Proceedings*, 1991. APF authorization and more.

Appendix A. Programming Examples

A.1 Loading a PC routine

```
* Get module length via BLDL, obtain pageable ECSA storage in that
* amount, load the module to that storage.
* The example assumes PC routines defined with AMODE(31).
         LA    R0,1               load number of entries for BLDL
         STH   R0,BLDLFF            and save it
         LA    R0,PDSBCLN         load len of basic PDS entry section
         STH   R0,BLDLLL            and save it
         ...
         LA    R9,1               set loop index
ETDEFLP  DS    0H                 R9 = 1 to 8
         LR    R15,R9             copy loop index,
         BCTR  R15,0                subtract 1
         SLL   R15,3                and multiply by 8
         LA    R15,PCRTNLST(R15)  point to current routine name
         MVC   PDS2NAME,0(R15)    copy routine name
         BLDL  PCRTNDCB,          DCB of load library already opened   +
               BLDLLIST
         LTR   R15,R15            OK?
         BZ    GETLDLN1           if yes, skip
         ...                      module not found
GETLDLN1 DS    0H
         SLR   R4,R4
         ICM   R4,7,PDS2STOR      get load module's length from BLDL
         STORAGE OBTAIN,          get storage                          +
               LENGTH=(R4),                                           +
               KEY=0,               in storage key 0                  +
               BNDRY=PAGE,          page-aligned for debugging ease   +
               SP=241,              pageable ECSA, not fetch-protected +
               LOC=(ANY,ANY)        anywhere
         LR    R2,R1              copy addr of storage
         ...                      save addr of storage
         XC    LOADLST,LOADLST    clear list
         LOAD  EPLOC=PDS2NAME,    load routine into global storage     +
               ADDR=(R2),           at this address                   +
               DCB=PCRTNDCB,        use DCB                           +
               SF=(E,LOADLST)
         ...                      do ETDEFs
* Names of PC routine load modules
PCRTNLST DS    0C
PCRTN0NM DC    CL8'COPPCX00'      EX=0
PCRTN1NM DC    CL8'COPPCX01'      EX=1
PCRTN2NM DC    CL8'PCRTNX02'      EX=2
PCRTN3NM DC    CL8'COPPCX03'      EX=3
PCRTN4NM DC    CL8'COPPCX04'      EX=4
PCRTN5NM DC    CL8'COPPCX05'      EX=5
PCRTN6NM DC    CL8'COPPCX06'      EX=6
PCRTN7NM DC    CL8'COPPCX07'      EX=7
PCRTNNBR EQU   (*-PCRTNLST)/8     number of PC routines
PCRTNBAS DCB   DDNAME=PCRTNLIB,DSORG=PO,MACRF=E
PCRTNBAL EQU   *-PCRTNBAS         length of static DCB
```

A.2 Mapping DSECTs

Here are DSECTs for various fields in the examples. The first is for working storage.

```
WORKAREA DSECT
         ...
SAVESSVT DS    A                 addr of HRLM's SSVT
LXL      DS    0F
LXCOUNT  DS    F                 number of LXs requested
LXVALUE  DS    F                 LX from LXRES routine or HSSVTLX2
LXL2     DS    0F
LXCOUNT2 DS    F                 number of LXs
LXVALUE2 DS    F                 LX from LXRES routine
TKL      DS    0F
TKCOUNT  DS    F                 number of ETs requested
TKVALUE  DS    F                 token created by ETCRE service
AXL      DS    0F
AXCOUNT  DS    H                 number of AXs
AXVALUE  DS    H                 AX value
         DS    0F
PCRTNDCB DS    XL(PCRTNBAL)      DCB copied from static storage
         DS    0F
BLDLLIST DS    0XL4
BLDLFF   DS    XL2               number of entries
BLDLLL   DS    XL2               length of each entry
         IHAPDS DSECT=NO
         DS    0F
LOADLST  LOAD  0,SF=L
LOADLSTL EQU   *-LOADLST         length of list
         DS    0F
ETDLIST  DS    XL(ETDLISTL)      entry table description list
ETCONLST ETCON MF=L
LXRESLST LXRES MF=L
LXRESLSL EQU   *-LXRESLST        length of list
LXFRELST LXFRE MF=L
LXFRELSL EQU   *-LXFRELST        length of list
ETDESLST ETDES MF=L
ETDESLSL EQU   *-ETDESLST        length of list
SAVECR4  DS    F                 CR4 of ADMRTN
TARGPASN DS    H                 PASN of caller address space
SRVRPASN DS    H                 PASN of server address space
SAVEKEY  DS    X
* DSECTs and EQUs
CR4      EQU   4                 EQU for control register 4
         YREGS ,                 register EQUs
         IHAETD FORMAT=1         entry-table entry
         CVT   DSECT=YES
         IEFJESCT ,              JES communication table
         IEFJSCVT ,              subsystem CVT
         IEFJSSVT ,              subsystem vector table
*************************************************************************
*                                                                       *
* Subsystem vector table for HRLM:  Further mapping of IEFJSSVT.        *
*                                                                       *
*************************************************************************
SSVT     DSECT                   continuation of SSVT
HSSVTLX2 DS    F                 system LX
HSSVTLL2 DS    F                 non-system LX
HSSVTET2 DS    F                 token of created entry table
HSSVTAX2 DS    H                 AX
HSSVTLEN EQU   *-SSVT            length of HRLM's SSVT + extension
```

Index

V