Concepts of Enterprise Systems Architecture/370

by K. E. Plambeck

Enterprise Systems Architecture/370™ (ESA/370™) is the next step in the architectural evolution of IBM's large processors from System/360 to System/370 to System/370 Extended Architecture (370-XA). ESA/370 includes all of the facilities of 370-XA and also significant new facilities. It greatly increases the amount of apparent main storage that is readily available for use. It provides for more efficient secure program linkage, with increased status saving and restoring, among hierarchically or nonhierarchically organized programs. It allows improved control program efficiency.

The foremost central-processor-related attribute of the architectural progression in the Enterprise Systems Architecture/370™ (ESA/370™) family has been ever-increasing apparent main storage. This storage was expanded first through the mechanism of virtual storage, as provided by dynamic address translation in System/370,[1] and then through the increase of the address size from 24 bits to 31 bits in System/370 Extended Architecture (370-XA).[2]

Associated with the increase in apparent main storage (referred to hereafter simply as storage), the means for providing *domains* of storage has been augmented, where a domain is defined abstractly as "a set of information and authorizations for the manipulation of that information within a computing system."[3] Domains were initially provided by storage keys associated with 2048-byte (now 4096-byte or 4K) blocks of storage and a matching access key, called the PSW key, in the program status word (PSW) (and also by the privileged supervisor state, which is used to establish a domain and can be used to circumvent any type of domain). The domains were then enhanced by the inherent ability of dynamic address translation (assuming appropriate support by a control program) to provide not only a virtual storage but a unique *address space* for each user of the system; that is, a set of contiguous virtual addresses that each user may use but which maps, at least partially, to actual main storage that is separately assigned to each user. The segment-protection facility of System/370 and the page-protection facility of 370-XA are other additions to the original domain-producing facilities.

Domains are a way of isolating users, but complete isolation is usually not desired. Accordingly, the *dual address-space* (DAS) facility was added as a feature to System/370 in 1980 and then made standard in 370-XA. DAS improves the usability of user domains by allowing machine-effected, as opposed to operating-system-effected, data access and program linkage between domains.

ESA/370[4] further enhances storage amounts and user domains. Specifically, ESA/370 provides improved means for

- Moving data between two address spaces
- Using the complete instruction set to operate on data in other than the instruction address space
- Transferring control between address spaces

The new facilities of ESA/370 evolved from DAS, and an understanding of DAS is a precursor to understanding the new facilities. In this paper, the way DAS is used to communicate between address spaces and the problems that still remain when DAS is used are described, as are the expanded and new solutions provided by ESA/370.

There is a prefix register for each central processor in a multiprocessing configuration.

The functions provided by DAS and by the new facilities of ESA/370 are under the control of authorization elements and mechanisms. The latter are described along with the functions and are summarized in the Appendix.

We first provide background with brief discussions of storage terminology and dynamic address translation. Information about the general attributes and development of System/360, System/370, and 370-XA is given in References 5, 6, 7, and 8.

Storage. Main storage, which is sometimes called processor storage, is "the program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing."[9] In the early members of the ESA/370 family of machines, main storage consisted of magnetic cores until the advent of transistorized storage in the System/370 Model 145 in 1970.

In a virtual-storage system, main storage is called real storage. Virtual storage is "the storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses."[9]

The term "apparent main storage" is used in this paper to mean main storage in a nonvirtual-storage system and virtual storage in a virtual-storage system.

In ESA/370, virtual storage is provided in 4096-byte units called pages, and the 4096-byte unit of real storage that corresponds to a page, if any, is called a page frame. Pages are contained within 1-megabyte units called segments. Pages, page frames, and segments begin on integral boundaries; that is, the address of the first location within a unit is a multiple of the size of the unit.

An object is usually said to reside in real storage only if it is addressed by means of a real address. (Most of the control structures specified in the architecture point to one another by means of real addresses and so are said to be in real storage.)

Absolute storage is the same as real storage in that, in general, real addresses correspond one-to-one to absolute addresses. The exception is real addresses 0 to 4095, which are said to designate locations within the prefix area and are changed to absolute addresses by adding the contents of a prefix register to them.

There is a prefix register for each central processor in a multiprocessing (shared main storage) configuration (as well as in a uniprocessor configuration). The contents of the prefix register are called the prefix, and the process of adding the prefix to real addresses 0 to 4095 is called prefixing. The prefix designates a 4096-byte block (on an integral boundary) of absolute storage.

The prefix area contains storage locations assigned for special use by the central processor. For example, the prefix area contains the locations in which the old PSW is stored and from which the new PSW is fetched during an interruption. Prefixing allows each central processor to have its own prefix area in absolute storage and thereby avoids interference with the other central processors in the configuration. (The assigned storage locations are actually in only the first 512 bytes of the prefix area, although this is subject to change. There is also reverse prefixing, in which the real-storage block designated by the prefix is mapped to absolute-storage block 0. Reverse prefixing allows all central processors to access absolute-storage block 0.)

A seven-bit storage key is associated with each 4096-byte block of absolute storage. The storage key consists of four access-control bits, a fetch-protection bit, a reference bit, and a change bit. A store access to the block is prohibited unless a four-bit access key either matches the access-control bits or is all zeros. For central-processor operations (as opposed to I/O

operations), the access key is the PSW key in the PSW. When the fetch-protection bit is one, a fetch access is similarly prohibited. The reference and change bits are set to one when a reference is made to the block and when the contents of the block are changed, respectively. Privileged instructions are provided for setting and inspecting the storage key. The protection provided by means of the storage key is called key-controlled protection.

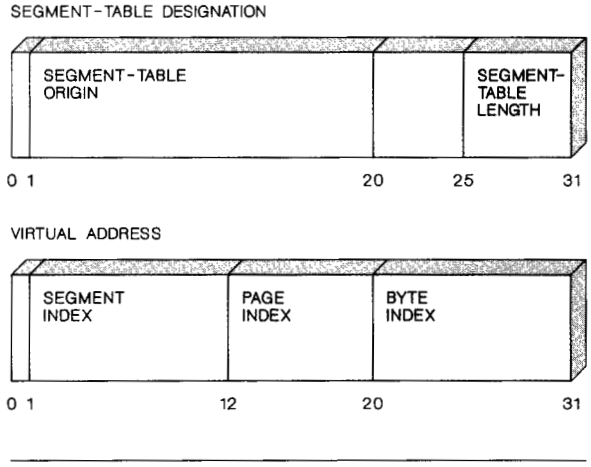
Auxiliary storage is "a storage device that is not main storage,"[9] for example, magnetic tape or a direct-access storage device (DASD). (The IBM 3090 processor expanded storage, which is addressable by block number instead of by byte address, is a form of auxiliary storage.)

Dynamic address translation. In ESA/370, a virtual address is translated to a real address by a process called dynamic address translation (DAT). DAT uses a *segment-table designation* in a control register and a segment table and a set of page tables in real or absolute storage. (It cannot be predicted whether prefixing is applied to references to the DAT tables during DAT. This simply means that no part of the DAT tables should be placed in the prefix area or absolute-storage block 0.)

The segment-table designation and a 31-bit virtual address are shown in Figure 1. The segment-table designation specifies the origin and length of the segment table. The segment-index part of the virtual address selects an entry in the segment table, which entry, if valid, specifies the origin and length of a page table. The page-index part of the virtual address selects an entry in the designated page table. The page-table entry, if valid, contains the address of a page frame in real storage. This page-frame address, with the 12-bit byte-index part of the virtual address appended on its right, is the resulting real address. An interruption occurs if the segment-table entry or page-table entry is invalid, with the result that the control program normally makes the entry valid and then causes the translation to be repeated.

The control program normally marks a segment-table entry as invalid if it has not yet created the page table that is to be pointed to from the segment-table entry, and it normally marks a page-table entry as invalid if it has not yet assigned a page frame to correspond to the page represented by the page-table entry. When the control program assigns a page frame after the first reference to a page, the page frame normally contains all zeros. If it is not the first

Figure 1 Segment-table designation and virtual address



reference, the control program reads the contents of the page from auxiliary storage and places them in the page frame. The control program reclaims page frames by writing their contents to auxiliary storage if the contents have been changed (as indicated by a change bit).

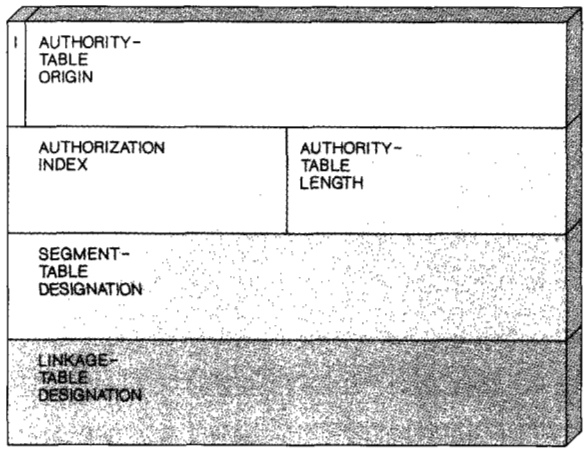
A segment-table designation corresponds to an address space. The control program provides addressability to different address spaces by changing the segment-table designation in the control register.

In addition to the protection against stores and fetches provided by the storage key, stores can be prohibited in System/370 by means of a bit in the segment-table entry and in 370-XA and ESA/370 by means of a bit in the page-table entry.

The first and third of the following three subsections describe the situation prior to ESA/370 and MVS/ESA™. Changes are described later in this paper.

Common and private segments. It is often useful to have a segment containing the same information exist at the same location in all address spaces as, for example, a segment containing part of the control program. Such a segment is called a common segment. A common segment is provided by having each segment table contain, at the same segment-index position, the designation of the same page table. A segment that is not a common segment is called a private segment.

Figure 2 ASN-second-table entry



In MVS, the set of all common segments in an address space is called the common area, and the set of all private segments is called the private area. Note that a program or an item of data present in the common area is in all address spaces.

Translation-lookaside buffer. The central processor uses a translation-lookaside buffer (TLB) to carry out DAT with good performance. The TLB contains copies of DAT-table entries that have been used recently to perform translations. If the same DAT-table entries are required again, the copies of them in the TLB are used instead of fetching the entries from real or absolute storage again. Except as will be described, a TLB entry can be used for a translation only if the segment-table origin that was used when the entry was formed (this origin is kept as part of the entry) is the same as the segment-table origin that is part of the current segment-table designation in the control register. This condition ensures that a TLB entry formed because of a reference to one address space will not be used to translate a reference to another address space.

Common-segment bit. To allow a still further improvement in performance, a segment-table entry contains a bit named the common-segment bit. This bit should be set to one in each segment-table entry that defines a common segment. A TLB entry formed from a segment-table entry in which the common-segment bit is one, or from any page-table entry designated by that segment-table entry, is used to perform a translation regardless of the segment-table origin currently being used. Thus, only one set of TLB entries is needed for a common segment regardless of how many different address spaces contain the common segment, and having one set decreases the number of references to the DAT tables in real or absolute storage.

Dual address-space facility

The dual address-space (DAS) facility allows operations on two address spaces concurrently. These address spaces are called the primary address space and the secondary address space, and they are defined by a *primary segment-table designation* (PSTD) and a *secondary segment-table designation* (SSTD) in control registers.

All address spaces are identified by means of a 16-bit binary *address-space number* (ASN). The ASNs for the current primary and secondary address spaces are also in control registers.

DAS includes instructions that use a two-level table structure, anchored in a control register, to translate an ASN and locate the corresponding *ASN-second-table entry* (ASTE). The ASN-second-table entry resides in real storage and is the principal control structure representing an address space. It contains the segment-table designation that defines the space. It also contains other control information that applies to the space, as shown in Figure 2. This other information is described in the subsections that follow.

The invalid (I) bit in the ASN-second-table entry indicates, when one, that the entry is invalid; that is, either (1) the entry is not currently assigned to represent an address space, or (2) the segment and page tables for the address space are not currently in real or absolute storage.

The instructions that perform ASN translation obtain the segment-table designation from the located ASN-second-table entry and place it in a control register as the primary or secondary segment-table designation. Such an instruction also updates the corresponding ASN in a control register.

DAS allows the following:

- A program being executed in the primary address space can move data between the primary address space and the secondary address space.
- A program being executed in the common area (the area that is in all address spaces) can use the complete instruction set to operate on data in

either the primary address space or the secondary address space. (However, as will be described, it must issue a control instruction to select which one of these two spaces it can access.)

- A program being executed in one address space can transfer control to a program in another address space. The address space containing the called program becomes the primary address space, and the space containing the calling program becomes the secondary address space. This setting of the primary and secondary address spaces allows the called program to move data between the address spaces. There is also a return linkage mechanism.

We now describe the above capabilities in more detail and point out problems that still remain. The authority elements and mechanisms that control the capabilities also are described.

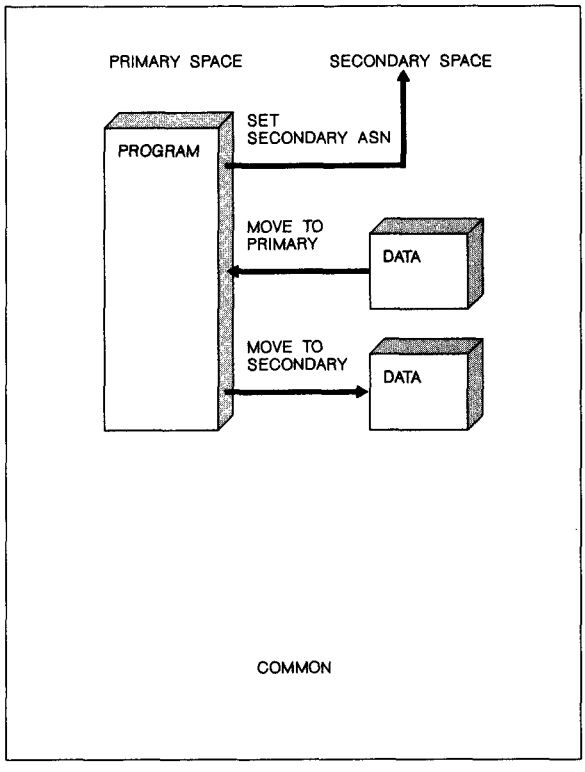
Moving data between spaces with DAS. DAS includes the following instructions, whose operations are illustrated in Figure 3:

- MOVE TO PRIMARY, which moves data from the secondary address space to the primary address space
- MOVE TO SECONDARY, which moves data from the primary address space to the secondary address space
- SET SECONDARY ASN, which causes an address space designated by means of its ASN to become the secondary address space

When setting the secondary address space to other than the current primary address space, SET SECONDARY ASN must be authorized by an *authorization index* (AX), which is a 16-bit binary integer in a control register. Each address space has associated with it an *authority table* that contains a list of the authorization indexes that allow the space to be established as the secondary space by means of the SET SECONDARY ASN instruction.

Each address space has associated with it a single authorization index that is in the ASN-second-table entry for the space. The authorization index for an address space is placed in the authorization-index control-register position whenever the space becomes the primary address space. This authorization index then applies to the programs that are executed in the primary address space. The origin and length of the authority table for an address space also are in the ASN-second-table entry for that space.

Figure 3 Moving data between the primary and secondary address spaces



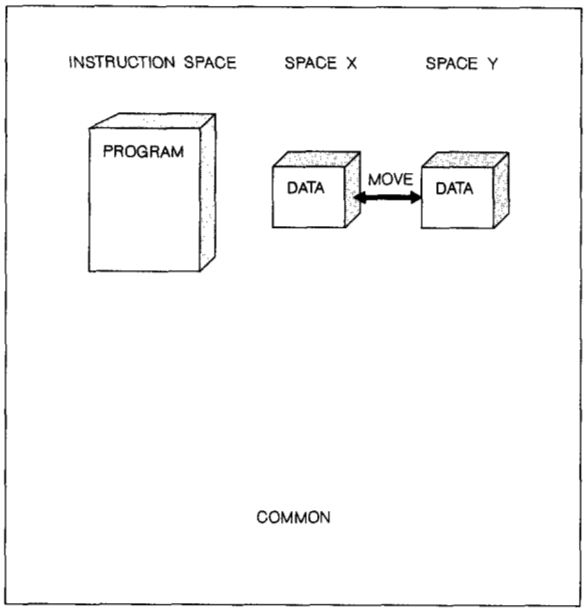
MOVE TO PRIMARY and MOVE TO SECONDARY have an operand that specifies the key to be used, instead of the PSW key, to access the secondary address space; the PSW key is used to access the primary address space. This operand is useful when the source and target data areas have different storage keys. (For MOVE TO PRIMARY, it is useful when the source data area is fetch-protected.)

The use of a particular secondary-space access key by MOVE TO PRIMARY and MOVE TO SECONDARY must be authorized by the *PSW-key mask* (PKM), a 16-bit mask in a control register. An access key is authorized if the bit corresponding to it in the PSW-key mask is one.

DAS also includes these instructions:

- MOVE WITH KEY, which moves data only within a single address space but, in the same way as MOVE TO PRIMARY (authorized by the PSW-key mask), uses a specified access key, instead of the PSW key, to access the source data area

Figure 4 Moving data between arbitrary spaces (not in DAS)



- SET PSW KEY FROM ADDRESS, which sets the PSW key to a specified value if this value is authorized by the PSW-key mask

The PSW-key mask that applies to a program is a property of the entry point at which the program is given control; unlike the authorization index, it is not a property of the primary address space in which the program is executed. (How a PSW-key-mask value is assigned to a program is described later.)

Problem—Moving data between arbitrary address spaces. A program issuing the instructions MOVE TO PRIMARY or MOVE TO SECONDARY can move data between two address spaces only when one of the spaces is the instruction space; data cannot be moved between two arbitrary address spaces. Figure 4 illustrates this missing capability.

Problem—Applicability of authorization index. Because all of the programs in a particular primary address space have the same authorization index, all of them are equally authorized to establish another address space as the secondary one by means of the SET SECONDARY ASN instruction. This problem could be circumvented by mapping two or more address spaces (different ASNs) to real storage by means of exactly the same set of DAT tables and associating a different authorization index with each space. However, this circumvention would increase the overhead related to the management of address spaces by the control program.

Operating on data in another address space with DAS. DAS provides two virtual translation modes that are selected by a bit in the PSW, which has meaning only when DAT is on. The names and properties of these modes are

- *Primary-space mode.* Instruction addresses (the addresses of locations from which instructions are fetched, including branch addresses and the address of the target of the EXECUTE instruction) and data addresses (called logical addresses in the architecture) are in the primary address space.
- *Secondary-space mode.* Whether instruction addresses are in the primary address space or the secondary address space cannot be predicted. Data addresses are in the secondary address space.

The SET ADDRESS SPACE CONTROL instruction is used to switch between the primary-space and secondary-space modes. The INSERT ADDRESS SPACE CONTROL instruction is used to obtain an indication of the current mode so that the current mode can be saved and then restored later by SET ADDRESS SPACE CONTROL. No authorization is required for these instructions.[10]

The unpredictability of instruction addresses in the secondary-space mode means that first one instruction could be fetched from the primary address space and then the next instruction could be fetched from the secondary address space. The only way to have a program run reasonably under these conditions is for it to be in the common area. A program in the common area can switch between the primary-space mode and secondary-space mode and can thereby use the complete instruction set to operate on data in either the primary address space or the secondary address space, as shown in Figure 5.

The program in Figure 5 is initially in the primary-space mode and loads X from the primary address space into a register. It then issues SET ADDRESS SPACE CONTROL to change to the secondary-space mode and compares the contents of the register to Y in the secondary address space.

Problem—Use of common, and performance, in the secondary-space mode. A large problem is the requirement that a program using the secondary-space mode must be in the common area; placing the

program in the common area increases the size of the common area and decreases the size of the private area. Also, SET ADDRESS SPACE CONTROL is a serializing instruction (drains the pipeline by waiting until all stores are completed and by causing all prefetched information to be discarded), so it is not practical, if good performance is desired, to switch back and forth frequently between the primary-space and secondary-space modes.

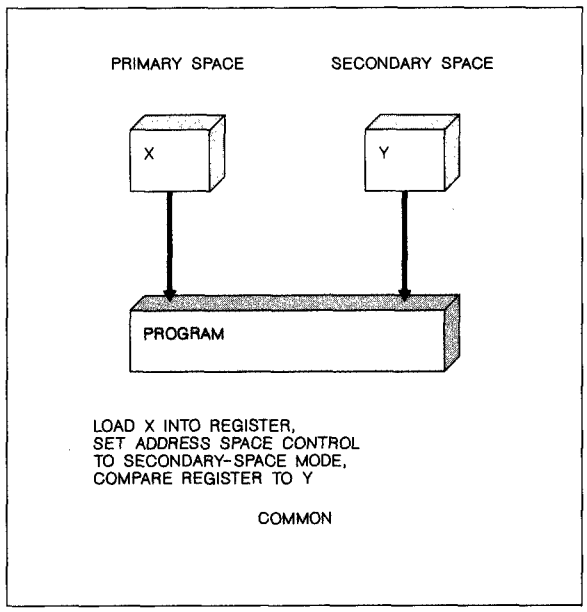
Some history is in order here. The secondary-space mode was originally defined so that, in it, instruction addresses would be predictably in the primary address space. This definition does not require a program using the mode to be in the common area. However, neither the IBM 3031 and 3033 processors nor the 308X processor, which were the first machines to implement DAS, have more than one segment-table designation in hardware. Therefore, those processors fetch instructions from the secondary address space when in the secondary-space mode.[11] This is undesirable because it means that SET ADDRESS SPACE CONTROL is, in effect, a branch instruction; the next instruction is not fetched from the next sequential location unless the program is in the common area.

The IBM 3090 processor does have two segment-table designations in hardware and does fetch instructions from the primary address space when in the secondary-space mode. To provide compatibility among the 3031, 3033, 308X, and 3090, the "unpredictable" definition was introduced.

At one time it was planned to announce an "instruction-space-separation facility" (ISSF) on the 3090 to explain its different operation in the secondary-space mode. However, it was finally realized that the originally intended definition of the secondary-space mode (the same definition as ISSF) is unsatisfactory because most programs are always written to include some data, if only constants, within the programs. Although the primary purpose of a program may be to access only the secondary address space, the program usually must actually access both the primary address space and the secondary address space in rapid succession, and this is not possible with good performance using DAS.

In summary, the mechanism of a secondary address space, with respect to which the MOVE TO PRIMARY and MOVE TO SECONDARY instructions can be used, is very useful, but the mechanism of a secondary-space mode is not.

Figure 5 Operating on data in two address spaces



DAS program linkage. DAS includes the PROGRAM CALL and PROGRAM TRANSFER instructions for transferring control between programs in either the same or different address spaces. Generally, PROGRAM CALL is used to perform a calling linkage, and PROGRAM TRANSFER is used to perform the related return linkage.[12]

Calling linkage. PROGRAM CALL uses an operand that is a 20-bit binary integer called a *PC number.* PROGRAM CALL translates the PC number to locate an *entry-table entry* that specifies state changes to be made, as follows:

- *Primary address space.* The primary address space can either be left unchanged, with the operation then called PROGRAM CALL to current primary, or be changed to one specified by means of its ASN in the entry-table entry, called PROGRAM CALL with space switching. In a space-switching operation, the ASN is translated to obtain a segment-table designation, which replaces the primary segment-table designation.
- *Instruction address.* The instruction address in the PSW is replaced from the entry-table entry.
- *Problem state.* The problem-state bit in the PSW is replaced from the entry-table entry, thus placing the machine in either the problem state or the supervisor state.

- *Addressing mode.* The addressing-mode bit in the PSW is replaced from the entry-table entry, thus placing the machine in either the 24-bit addressing mode or the 31-bit addressing mode.
- *PSW-key mask.* The PSW-key mask is ORed with an *entry key mask* (EKM) in the entry-table entry, and the result replaces the PSW-key mask. This action can increase the authority provided by the PSW-key mask.[13]

PC-number translation locates an entry-table entry within a two-level table structure consisting of a *linkage table* and of entry tables designated by linkage-table entries. The origin and length of the linkage table, called the *linkage-table designation* (LTD), is in a control register. The linkage table and entry tables are in real storage.

PROGRAM CALL sets the secondary address space equal to the old primary address space: The old primary segment-table designation replaces the secondary segment-table designation. This occurs even in a to-current-primary operation so that the called program will be unaware of whether the linkage was space-switching and can always access its caller's parameters in the secondary address space.

In the case of a space-switching PROGRAM CALL, a new authorization index and a new linkage-table designation are obtained from the ASN-second-table entry for the new primary address space and are placed in their control-register positions.

The PROGRAM CALL operation must be authorized by means of a 16-bit *authorization key mask* (AKM) in the entry-table entry. The authorization key mask is ANDed with the PSW-key mask before the PSW-key mask is replaced. If the result is nonzero, the operation is authorized.

Because all of the programs being executed in a particular primary address space have the same linkage-table designation in a control register, all of them have available the same set of entry-table entries. However, because the programs can have different PSW-key masks and the entry-table entries can contain different authorization key masks, the programs can be authorized differently to use the entry-table entries in PROGRAM CALL operations. For example, with the assumption that a primary address space contains a user program and parts of subsystem A and subsystem B, the user program might be authorized to call subsystem A but not subsystem B, whereas subsystem A is authorized to call subsystem B.

Return linkage. PROGRAM CALL places the old primary ASN, problem-state bit, addressing-mode bit, PSW-key mask, and the return address, in general registers. The called program is expected to save the contents of these registers and then provide the contents as operands of the PROGRAM TRANSFER instruction that it uses to return to the calling program.

If the ASN specified as an operand of PROGRAM TRANSFER is the same as the current primary ASN, the operation is called PROGRAM TRANSFER to current primary; otherwise, it is called PROGRAM TRANSFER with space switching.

The PROGRAM TRANSFER instruction performs operations as follows:

- If the operation is space-switching, the ASN that is specified as an operand is translated to obtain a segment-table designation, which replaces the primary segment-table designation. This must be authorized by the current authorization index (the authorization index for the address space of the called program). The authority table of each address space contains a list of the authorization indexes that allow the space to be established as the primary address space by means of PROGRAM TRANSFER. This list is different from the list that applies to SET SECONDARY ASN.[14] If the operation is to-current-primary, the current primary segment-table designation remains unchanged.
- The secondary segment-table designation is set equal to what is now the primary segment-table designation.
- The problem-state bit, addressing-mode bit, and return address that are specified as operands are placed in the PSW. However, the problem-state bit is allowed to specify the supervisor state only if PROGRAM TRANSFER is executed in the supervisor state.
- The PSW-key mask (in a control register) is ANDed with the PSW-key mask that is specified as an operand, and the result replaces the PSW-key mask. This action can decrease the authority provided by the PSW-key mask.

A space-switching PROGRAM TRANSFER places a new authorization index and linkage-table designation in their control-register positions—both are obtained from the ASN-second-table entry for the returned-to primary address space.

Problem—Register save areas. The conventional procedure when transferring control through a chain

of programs is for each program to provide a save area pointed to by general register 13, and then for each called program to save the general-register contents in the caller's save area and forward-chain and backward-chain the caller's save area and its own save area together. Then if an error occurs requiring analysis of a dump, the chain of save areas can be examined in either the forward direction, starting at a standard location, or the backward direction, starting at the location designated by register 13.

The code that a called program usually uses to manipulate save areas is inadequate when it and the calling program are in different address spaces. This is particularly true when the called program is re-entrant and would normally begin by saving registers in the caller's save area before using a control program service to obtain storage of its own. Therefore, when PROGRAM CALL was introduced, so was an MVS control program service named PCLINK. PCLINK establishes a special save area and backward-chains it to the caller's save area. Because PCLINK must disable the machine for I/O and external interruptions under certain circumstances, it requires that the program requesting the PCLINK service be in the supervisor state.

It is unreasonable for different subsystems to be hierarchically ranked.

Two problems are related to PCLINK. First, although it was intended that PROGRAM CALL be a fast means of going across address spaces without intervention by the control program, serviceability requirements and functional requirements of re-entrant programs cause the called program to immediately invoke the PCLINK service, which detracts from the performance.[15] Second, since PCLINK requires the supervisor state, PROGRAM CALL cannot be used to give control to a problem-state program in another address space.

Ideally, subsystems should be problem-state programs so that they do not pose even a nonmalicious threat to system integrity.

Problem—Hierarchical linkage. PROGRAM CALL provides only a hierarchical-type linkage; that is, the called program is always of equal or higher authority than the calling program. This hierarchy is evidenced in three ways:

1. PROGRAM CALL cannot be used to give control from a supervisor-state program to a problem-state program because the paired PROGRAM TRANSFER instruction cannot restore the supervisor state when executed in the problem state.
2. PROGRAM CALL can increase the authority provided by the PSW-key mask or leave it unchanged. It cannot decrease the authority, nor can it substitute a completely different authority.
3. Because PROGRAM CALL sets the secondary segment-table designation equal to the old primary segment-table designation, the called program always has access to the instruction address space of the calling program.

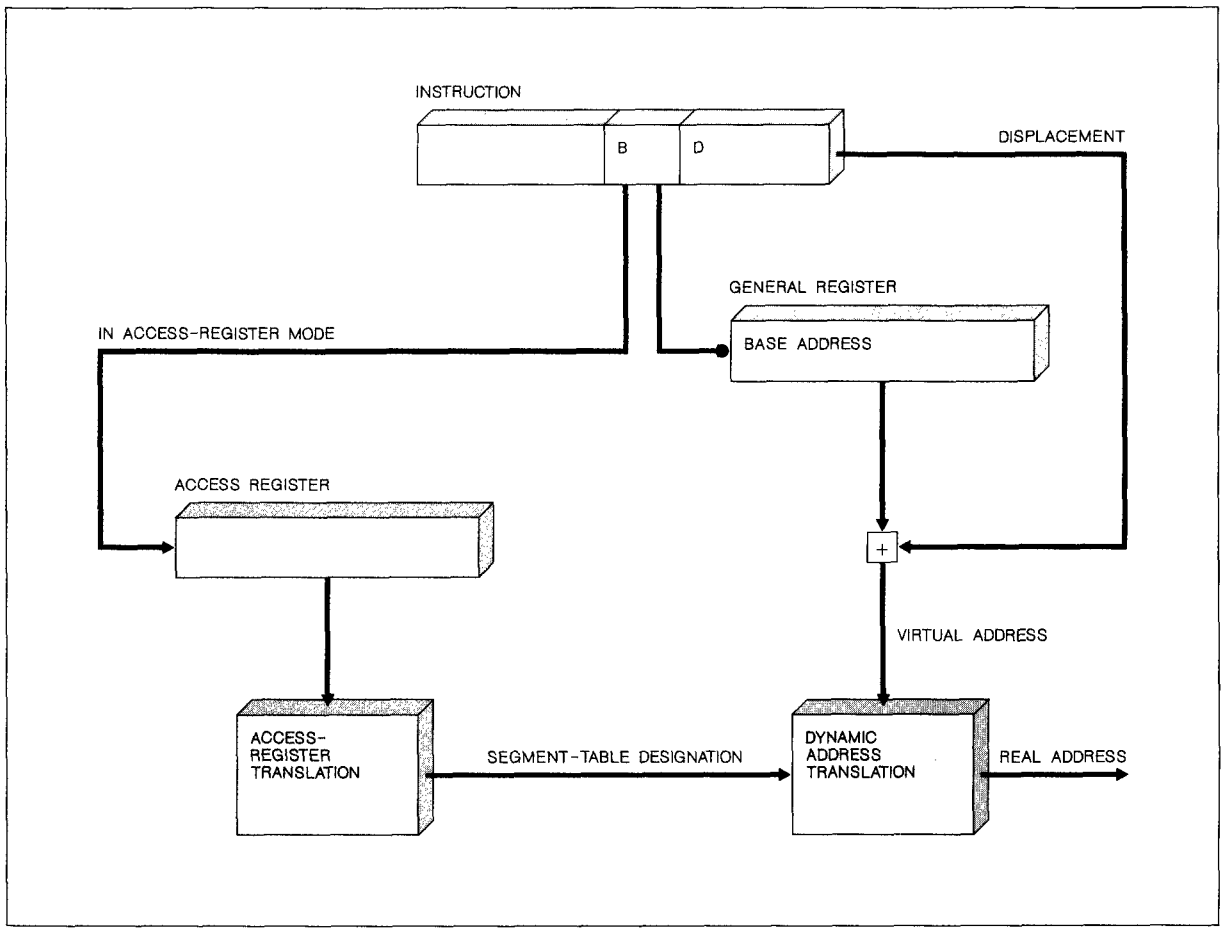
Although it may usually be reasonable for a subsystem to have higher authority than a user program, it is unreasonable for different subsystems to be hierarchically ranked. For example, to do distributed data processing, a database subsystem calls a terminal subsystem to send data to another node in the teleprocessing network, which, at the other node, causes the terminal subsystem to call the database subsystem. Neither of these subsystems should have to expose its resources to the other one.

The use of PROGRAM TRANSFER to give control from a subsystem to a user exit routine and then of PROGRAM CALL to return to the subsystem (mentioned in a note previously) is symptomatic of the inability to use PROGRAM CALL to perform a nonhierarchical linkage.

Problem—Setting of secondary address space by PROGRAM TRANSFER. A minor problem is that PROGRAM TRANSFER, even a PROGRAM TRANSFER to the current primary, sets the secondary segment-table designation equal to the new primary segment-table designation. Thus, the returned-to program may no longer have the same secondary address space as it did before it issued PROGRAM CALL. If the returned-to program is authorized, it can use SET SECONDARY ASN to re-establish its original secondary address space.

Problem—Calling fetch-protected code. A final problem is that PROGRAM CALL does not change the PSW key and, therefore, cannot be used to give control to

Figure 6 Use of access registers



fetch-protected code that has a different storage key than does the calling program. This problem is of increasing significance as subsystems become more and more proprietary.

ESA/370 facilities

The advanced address-space facilities of ESA/370 offer improvements in two major areas:

- *Data accessing.* Data can be accessed concurrently by the program in up to 16 different address spaces, including the instruction space, without changing any control parameters. This facility is provided by means of 16 new registers named *access registers.* Still more address spaces can be accessed by changing the contents of the access registers.
- *Program linkage.* The contents of an entry-table entry are augmented to allow increased status changing during a PROGRAM CALL operation. A *linkage stack* is provided for saving status during the operation and for restoring it by means of a new instruction named PROGRAM RETURN. A new branch-type linkage also uses the linkage stack.

The ESA/370 functions outlined above solve the DAS problems listed earlier in this paper. They also have other valuable features, as will be described.

ESA/370 is an upward-compatible extension of 370-XA, although access registers are completely new. Programs written to use DAS, or even only pre-DAS facilities, will continue to be executed successfully on any ESA/370 model.

ESA/370 also provides additional functions for improving the efficiency of the control program, and it includes a private-space facility for excluding the common area from specified address spaces.

Access registers. ESA/370 provides sixteen 32-bit access registers numbered 0 to 15. Access registers are used to address storage operands in a new translation mode named the *access-register mode*, which results from new bit settings in the PSW.

In the access-register mode, an instruction B or R field that designates a general register containing a storage-operand address also designates the same-numbered access register. The contents of the access register are used in a process called *access-register translation* (ART) to obtain the segment-table designation that will be used to translate, by means of DAT, the storage-operand address. The use of access registers is shown in Figure 6.

For example, consider the following instruction, which moves the second operand, of length L, to the first-operand location:

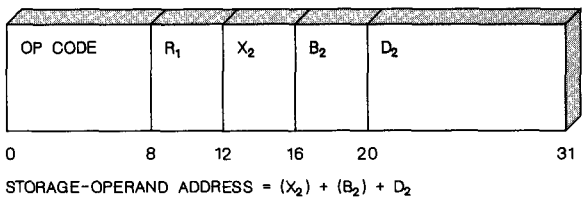
MVC 0(L,1),0(2)

General register 2 contains the address of the second operand, and general register 1 contains the address of the first-operand location. In the primary-space mode, both the second-operand and the first-operand location are in the primary address space (the instruction space). In the access-register mode, the second operand is in the address space specified by access register 2, and the first-operand location is in the address space specified by access register 1. Either or both of these spaces may be the same as or different from the primary address space, which is still the instruction space.

The DAS instruction SET ADDRESS SPACE CONTROL is changed so that it can set the access-register mode as well as the primary-space and secondary-space modes. INSERT ADDRESS SPACE CONTROL is changed so that it can return an indication of the access-register mode.

An address space specified by means of an access register is called an *AR-specified* address space. Access registers apply only to data addresses, not instruction addresses. In the access-register mode, instructions are always fetched from the primary address space. (It is not possible to branch from one address space to another.)

Figure 7 Format-RX instruction



When a B or R field designates access register 0, the contents of access register 0 are ignored, and the designated storage operand is treated as being in the primary address space.

A format-RX instruction has an X field that designates a general register containing an index that is to be added to the base address. The contents of the access register designated by the X field are ignored; only the access register designated by the B field is used in access-register translation. A format-RX instruction is illustrated in Figure 7.

Through the use of access registers, data can be moved between any two address spaces (see Figure 3), and the complete instruction set can be used to operate on data in multiple different address spaces without changing any control parameters.

The DAS instructions MOVE TO PRIMARY and MOVE TO SECONDARY are not allowed in the access-register mode. However, ESA/370 includes two new instructions: MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY. They can be used, either in or not in the access-register mode, to move data alternately in both directions between two storage areas that are fetch-protected by means of different storage keys, without changing the PSW key. Also, the DAS instruction MOVE WITH KEY remains available for use.

Access-list-entry token. The contents of an access register are called an *access-list-entry token* (ALET) because, in the general case, they designate an entry in a data area called an *access list*. Access-register translation uses the contents of the designated access-list entry to obtain the segment-table designation that will be used by DAT. The term "token" is used because an ALET does not directly convey any capability to access an address space; it only designates an access-list entry, which represents the actual capability.

ALETs are manipulable as ordinary data. ESA/370 includes instructions for transferring ALETs between access registers, general registers, and storage. Specifically, a called program can save the contents of the access registers in storage, load the access registers for its own purposes, and then restore the original contents so that the calling program will find them unchanged. An ALET can be transferred to and from access register 0 even though access register 0 does not participate in the addressing of a storage operand.

Entries in the access list are the addressing capabilities that are usable by means of access registers.

There are two special values of the ALET, 00000000 hex and 00000001 hex, referred to as ALET 0 and ALET 1, that specify the primary address space and secondary address space, respectively, without the use of an access-list entry. ALET 0 allows a program to have access to its own instruction space without the need to form an access-list entry that designates the space. After a space-switching PROGRAM CALL, ALET 1 similarly allows the called program to have access to the caller's instruction space. As will be described, a called program can be denied access to its caller's space.

Access list. Entries in the access list are the addressing capabilities that are usable by means of access registers. The access list is in real storage and is intended to be protected from the problem-state program to ensure the integrity of the addressing capabilities.

As described by Clark,[16] the control program MVS/ESA provides a service that allocates an access-list entry and returns an ALET designating the entry. The ALET can then be used by the requesting program to access the address space designated by the entry. The control program also provides a service for deallocating an access-list entry so that the entry can be reused.

An access-list entry is marked invalid when it is not in the allocated state. An exception is recognized on an attempt to use an invalid access-list entry.

Two access lists are actually available to a program at the same time. One is called the *dispatchable-unit* access list and the other the *primary-space* access list. The dispatchable-unit access list is intended to be permanently associated with the dispatchable unit (the task or process) on behalf of which the program is executed. The primary-space access list is a property of the primary address space in which the program is executed. A bit in the ALET specifies which one of the dispatchable-unit and primary-space access lists is designated by the ALET.

A bit in the access-list entry specifies whether the entry is *public* or *private.* No authorization is required for the use of a public access-list entry. The use of a private access-list entry must be authorized by an *extended authorization index* (EAX) in a control register. The extended authorization index may be a property of either the dispatchable unit or the program, as will be described. It is not a property of the primary address space in which the program is executed.

Through the use of the extended authorization index, an entry on a dispatchable-unit access list may be usable by some, but not all, of the programs that are executed to perform the work of the dispatchable unit. Similarly, an entry on a primary-space access list may be usable by some, but not all, of the programs that are executed in the corresponding primary address space. (In this discussion, a program is considered to be in the address space that is designated by the primary segment-table designation in a control register when the program is executed.)

The DAS authorization index has a bearing on the use of access registers since it authorizes the use of SET SECONDARY ASN in establishing a secondary address space, and the secondary address space can be accessed by means of ALET 1. As has been said, the authorization index *is* a property of the primary address space.

The length of an access-list entry is 16 bytes.

Access-list designation. The real origin and length of an access list are specified by an *access-list designation* (ALD). The access-list designation for the dispatchable-unit access list is in a control structure named the dispatchable-unit control table (DUCT).

The access-list designation for the primary-space access list is in an ASN-second-table entry named the primary ASTE. The primary ASTE is the ASN-second-table entry that represents the current primary address space. The real origins of both the dispatchable-unit control table and the primary ASTE are in control registers.

When the new facilities of ESA/370 are active (controlled by a control register bit), the address of the primary ASTE is in the control register which contains the linkage-table designation when the facilities are not active (and in DAS). When the new facilities are active, PROGRAM CALL obtains the linkage-table designation for use in PC-number translation from the primary ASTE, thus economizing on the number of control registers that are used.

The access-list-entry number designates an entry in the selected access list.

There are two formats of access-list designation, format 0 and format 1.

A format-0 ALD specifies the length of the access list in units of 128 bytes, thus making the length of the access list variable in units of eight 16-byte entries. With a format-0 ALD, the maximum length of the access list is 128 units of 128 bytes, which equals 16 kilobytes or 1024 entries.

A format-1 ALD specifies the length of the access list in units of 256 bytes, thus making the length of the access list variable in units of sixteen 16-byte entries. With a format-1 ALD, the maximum length of the access list is 256 units of 256 bytes, which equals 64 kilobytes or 4096 entries.

An ESA/370 model implements either the format-0 ALD or the format-1 ALD but not both; that is, the available ALD format is model-dependent, and no control is provided by which the program can specify the format. (As of this writing, all models implement format 0, and MVS/ESA supports an access list containing at most 256 entries.)

Since two access lists are available for use at any time and each access-list entry can specify a 2-gigabyte address space, 2048 entries can provide a total of 4 terabytes of addressable storage with the format-0 ALD, and 8192 entries can provide a total of 16 terabytes of addressable storage with the format-1 ALD.

Access-register translation. This subsection describes the access-list-entry token and the access-list entry in more detail and tells how they are used in the access-register-translation process to obtain a segment-table designation for use by DAT.

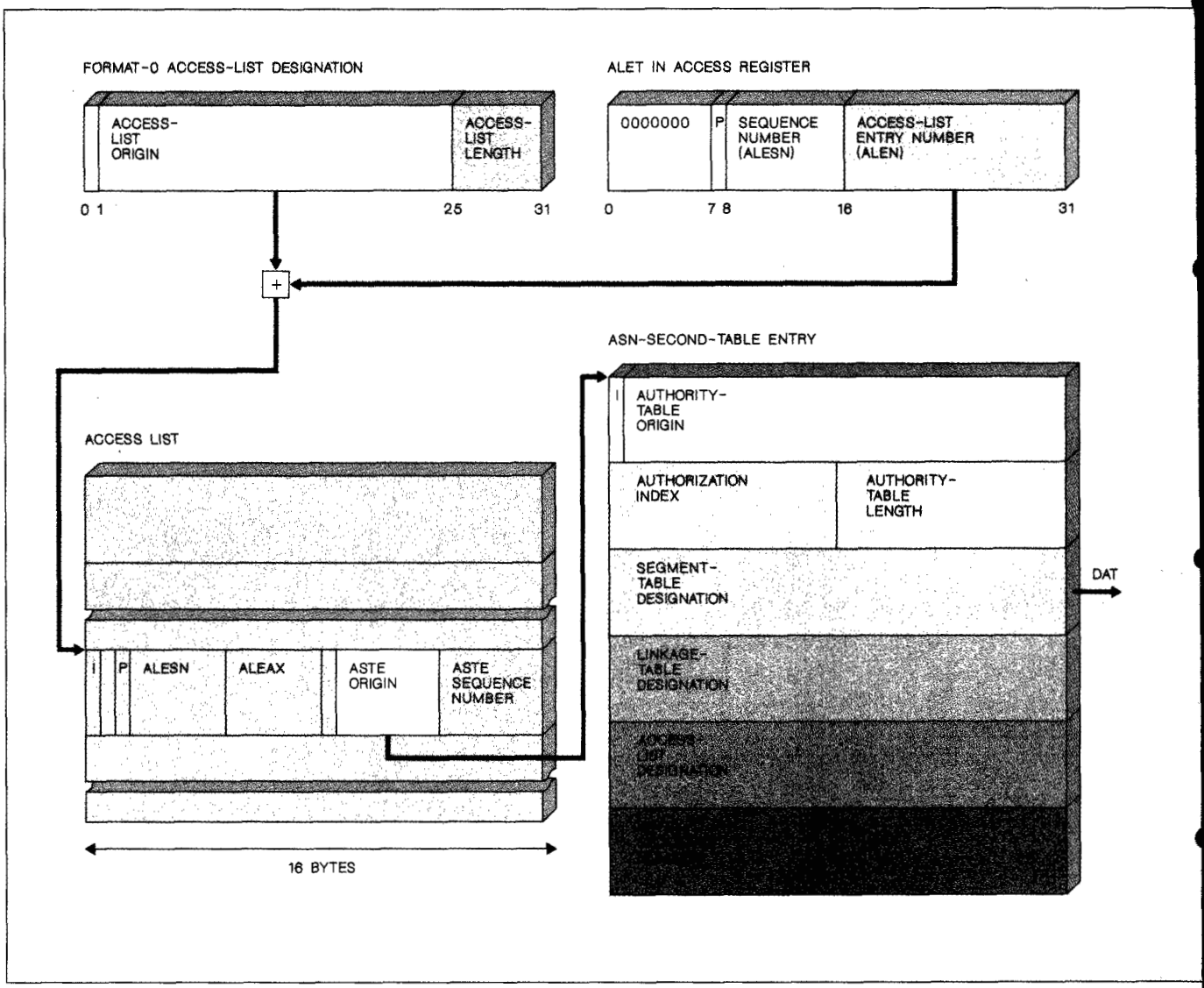
As shown in Figure 8, the ALET contains a 16-bit access-list-entry number (ALEN), an eight-bit sequence number (ALESN), and a primary-list (P) bit. The primary-list bit selects the dispatchable-unit access list if zero or the primary-space access list if one. The appropriate access-list designation, in either the dispatchable-unit control table or the primary ASTE, respectively, is then used in the access-register-translation process.

The access-list-entry number designates an entry in the selected access list. It is compared against the length field in the access-list designation to ensure that it designates an entry that is not beyond the end of the access list, and it is then added to the origin field in the access-list designation to form the real address of the designated access-list entry (all with appropriate shifting).

The invalid (I) bit in the access-list entry is tested for being zero.

The access-list-entry sequence-number (ALESN) field in the ALET is tested for being equal to the ALESN field in the access-list entry. This testing ensures that the ALET is not being used after the designated access-list entry was reallocated to specify a different address space. The control program increments the ALESN in the access-list entry each time it reallocates the entry, and it places the new value of the ALESN in the ALET that it returns to the program that requested the new allocation. Note that this is only a reliability mechanism. It provides no integrity since the requesting program can change the ALESN in the ALET to any value.

Figure 8 Access-register translation



The origin of the ASN-second-table entry that represents the address space specified by the access-list entry is obtained from the access-list entry, and the invalid bit in this ASTE is tested for being zero.

The 32-bit ASTE sequence number in the access-list entry is tested for being equal to the ASTE sequence number in the ASTE. This testing ensures that the access-list entry is not being used after a revocation of the addressing capability that the entry represents. The control program increments the ASTE sequence number in the ASTE each time it assigns the ASTE to represent a different address space and each time some authorization policy concerning the represented address space is changed, for example, each time a change is made to the authority table that is pointed to from the ASTE. When the control program allocates an access-list entry that provides access to an address space, it places the ASTE sequence number that is in the ASTE for the address space in the access-list entry. Note that this *is* an integrity mechanism. The mechanism is especially valuable because it

allows the control program to avoid keeping track of the access-list entries that designate each ASTE and then accessing those access-list entries in order to revoke the addressing capabilities that they represent.

If we assume that all tests have been passed so far—otherwise, an exception would have been recognized and a program interruption would occur—the authority of the program to use the access-list entry is tested in the following ordered steps:

1. If the private bit in the access-list entry is zero, the entry is public, and the program is authorized.
2. If the extended authorization index (EAX) in a control register equals the ALEAX in the access-list entry, the program is authorized.
3. If the EAX has a value which, if used as the authorization index (AX) would allow the SET SECONDARY ASN instruction to establish the specified address space as the secondary address space, the program is authorized.

If the program is authorized, the segment-table designation in the ASTE is provided to DAT; otherwise, an exception is recognized.

Note that an ASN is not used in access-register translation. If an ASTE designated by an access-list entry represents an address space containing only data—not also programs, so that its ASN need not be specified in an entry-table entry—the ASTE need not be in the two-level table structure that is indexed into by the ASN-translation process. Such an ASTE is sometimes called a pseudo ASTE. The number of possible pseudo ASTEs is not restricted by the number of possible ASNs (65 536).

When ALET 0 or ALET 1 is used, the process described above does not occur, and the primary segment-table designation or secondary segment-table designation is obtained for DAT. The primary segment-table designation is also obtained if access register 0 is used.

Access-register translation occurs on every storage-operand reference in the access-register mode. To improve performance, the machine implements an ART lookaside buffer (ALB) that is comparable to the translation-lookaside buffer (TLB) used by DAT.

ESA/370 program linkage. In ESA/370, PROGRAM CALL is changed to test a new bit, named the PC-type bit, in the entry-table entry. If this bit is zero, PROGRAM CALL performs the DAS operation, which is now called the basic operation. If the bit is one, PROGRAM CALL performs a new operation called the stacking operation. The stacking operation makes some state changes differently than the basic operation, and it saves the old state in an entry it forms in a linkage stack. A new instruction named PROGRAM RETURN logically deletes the linkage-stack state entry and restores the old state.

The linkage stack resides in virtual storage (in the home address space, which is described in a following section of this paper). The linkage stack can consist of one or more discontiguous sections, which are chained together by means of a header entry and a trailer entry in each section. The address of the last state entry in the linkage stack, or of the header entry of the current section if there is no state entry in that section, is in a control register.

It is intended that there be a separate linkage stack for each dispatchable unit and that the linkage stack be protected from direct manipulation by the dispatchable unit. There are instructions for extracting information from a state entry and for modifying one field in the entry.

A new branch-type instruction can be used to form a linkage-stack state entry. The branched-to program returns to its caller by means of the PROGRAM RETURN instruction.

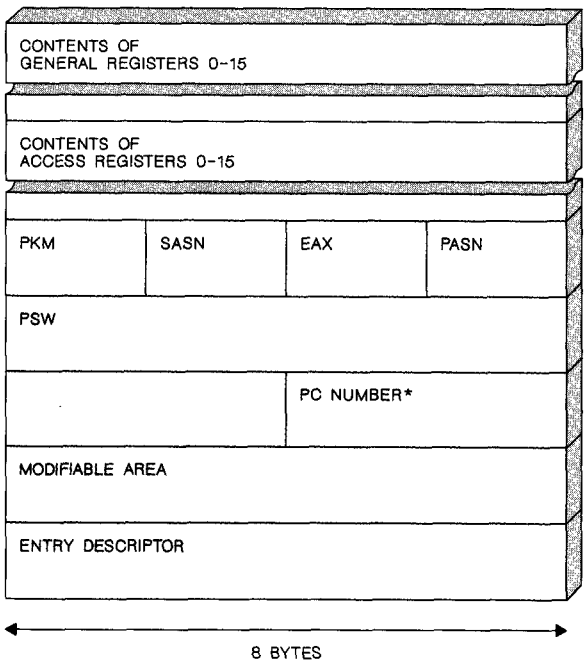
Key-controlled protection does not apply to the instructions that operate specifically on the linkage stack.

Calling linkage. Old programs that already contain PROGRAM CALL can have the stacking operation performed for them without the need to make any changes in the programs. All that needs to be done is to set the PC-type bit to one in the entry-table entry used and to provide some additional state information in the entry.[17]

The entry-table entry used in a stacking PROGRAM CALL operation specifies the following state changes:

- *Primary address space.* As in the basic PROGRAM CALL operation, either a to-current-primary or a space-switching operation can be performed.
- *Secondary address space.* Under the control of a bit in the entry-table entry, the secondary address space can be set equal to either the old primary address space or the new primary address space. By setting the secondary space equal to the new primary space, the called program is denied auto-

Figure 9 Linkage-stack program-call state entry



* BRANCH ADDRESS IN A BRANCH STATE ENTRY

matic access to the primary space of the calling program. The called program still may be able to establish that space as the secondary space by means of SET SECONDARY ASN.[18]

- *Instruction address, problem state, and addressing mode.* As in the basic operation, the instruction address, problem-state bit, and addressing-mode bit in the PSW are replaced from the entry-table entry.
- *PSW-key mask.* Under the control of a bit in the entry-table entry, the PSW-key mask is replaced by either (1) the OR of the PSW-key mask and the entry key mask (in the entry-table entry) or (2) the entry key mask. In the second case, the new PSW-key mask may provide reduced or entirely different authority.
- *PSW key.* Under the control of a bit in the entry-table entry, the PSW key can be either left unchanged or replaced from the entry-table entry.
- *Translation mode.* Under the control of a bit in the entry-table entry, the translation mode can be set to either the primary-space mode or the access-register mode.
- *Extended authorization index.* Under the control of a bit in the entry-table entry, the extended authorization index can be either left unchanged or replaced from the entry-table entry. Thus, the extended authorization index that authorizes use of access-list entries by the called program can be a property of either the dispatchable unit or the called program, respectively.

Stacking PROGRAM CALL, like basic PROGRAM CALL, is authorized by the authorization key mask in the entry-table entry. A space-switching stacking PROGRAM CALL instruction places a new primary-ASTE origin and authorization index (properties of the new primary address space) in the control registers. As illustrated in Figure 9, stacking PROGRAM CALL saves the following state information in the linkage-stack state entry that it forms:

- Complete PSW with an updated instruction address (the return address)
- Primary and secondary ASNs
- PSW-key mask and extended authorization index
- Contents of general registers 0-15 and access registers 0-15

Stacking PROGRAM CALL also places the PC number used, a two-word modifiable area, and an entry descriptor in the state entry.

An entry descriptor is in all linkage-stack entries. It contains an identification of the entry type (header, trailer, or state) and also a next-entry-size field that specifies the size of the next entry, if any. The entry-type code includes an indication of whether a state entry was formed by PROGRAM CALL or the new branch-type instruction. The next-entry-size field allows the logical end of the linkage stack to be determined in a storage dump.

ESA/370 includes instructions for extracting the information in the last state entry in the linkage stack and for modifying the contents of the modifiable area in the entry. The purpose of the modifiable area is to allow a program to "footprint" its progress so that appropriate recovery actions can be taken if the program fails.

A state entry formed by PROGRAM CALL is called a program-call state entry.

ESA/370 includes the BRANCH AND STACK instruction, which can be used in place of BRANCH AND LINK. The only state information changed by BRANCH AND

STACK is the instruction address in the PSW. BRANCH AND STACK forms a branch state entry that is the same as a program-call state entry, except that it indicates that it was formed by BRANCH AND STACK and contains the branch address instead of a PC number.

The addressing-mode bit and instruction address that are part of the complete PSW saved in a branch state entry can be either the updated[19] values in the PSW or can be specified in a register as an operand of BRANCH AND STACK. This register can be one that had link information placed in it by a BRANCH AND LINK, BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, or BRANCH AND SET MODE instruction. Thus, BRANCH AND STACK can be used either in a calling program or at (or near) the entry point of a called program, and, in either case, a PROGRAM RETURN at the end of the called program will return correctly to the calling program. The ability to use BRANCH AND STACK at an entry point allows the linkage stack to be used without changing old calling programs.

Return linkage. The ESA/370 instruction PROGRAM RETURN is used to return from a program given control by means of either stacking PROGRAM CALL or BRANCH AND STACK. PROGRAM RETURN logically deletes the last linkage-stack state entry, which may be either a program-call state entry or a branch state entry. If the last state entry is a program-call state entry, PROGRAM RETURN restores all of the state information that was saved in the entry, except that it leaves the contents of general registers 15, 0, and 1 and access registers 15, 0, and 1 unchanged—information can be returned to the calling program in these registers. If the last state entry is a branch state entry, PROGRAM RETURN restores only the complete PSW and the contents of general registers 2–14 and access registers 2–14. However, PROGRAM RETURN always leaves the PER mask in the PSW unchanged in order not to counteract a PER enablement or disablement that may have occurred while the called program was being executed.

A bit can be set to one in the entry descriptor of a linkage-stack state entry to cause a program interruption if PROGRAM RETURN operates on the entry. The control program may set this bit to one to guard against an erroneous use of PROGRAM RETURN, for example, when the last linkage instruction executed was a SUPERVISOR CALL instruction, in which case a return service of the control program should be used before PROGRAM RETURN.

Note that the combination of stacking PROGRAM CALL and PROGRAM RETURN permits nonhierarchical program linkage (linkage from a program with some amount of authority to a program with less or completely different authority).

Testing authorization. The ESA/370 instruction TEST ACCESS has as operands an access-list-entry token (ALET) in an access register and an extended authorization index (EAX) in a general register. TEST ACCESS applies access-register translation (ART) to the ALET, except that ART uses the EAX in the general register instead of the current EAX in a control register. TEST ACCESS indicates one of the following results: (1) the ALET is ALET 0, specifying the primary address space, (2) the ALET designates an entry in the dispatchable-unit access list and can be translated successfully by ART, (3) the ALET designates an entry in the primary-space access list and can be translated successfully by ART, or (4) the ALET is ALET 1, specifying the secondary address space, or causes ART to recognize an exception.

The principal purpose of TEST ACCESS is to allow a called program to determine whether an ALET passed to it by the calling program is authorized for use by the calling program by means of the EAX of the calling program. This purpose is in support of a possible programming convention in which a called program will not operate on an AR-specified address space by means of its own EAX unless the calling program is authorized to operate on the space by means of the EAX of the calling program. The called program can obtain the EAX of the calling program for use by TEST ACCESS by extracting it from the current linkage-stack state entry.

Another purpose of TEST ACCESS is to indicate when the ALET is ALET 0 or ALET 1. Because PROGRAM CALL may change the primary and secondary address spaces, ALETs 0 and 1 may specify different address spaces when used by the called program than when used by the calling program.

Still another purpose of TEST ACCESS is to indicate whether the ALET designates an entry in the primary-space access list, since the occurrence of such a designation after the primary address space has been changed by a space-switching program linkage is an error.

Control-program facilities. When MVS/ESA initiates a job step, which at least initially is a single dispatchable unit, it does so in an address space that is unique

Figure 10 Translation modes

PSW BIT		TRANSLATION MODE	INSTRUCTION ADDRESS SPACE	OPERAND ADDRESS SPACE
16	17			
0	0	PRIMARY-SPACE	PRIMARY	PRIMARY
0	1	ACCESS-REGISTER	PRIMARY	AR-SPECIFIED
1	0	SECONDARY-SPACE	PRIMARY	SECONDARY
1	1	HOME-SPACE	HOME	HOME

to the job step. This address space is called the home address space of the job step. MVS/ESA places the principal control blocks that represent the job step (for example, where status is saved when the job step is undispatched) in the home address space of the job step. If the job step uses PROGRAM CALL to give control to another address space and an I/O or external interruption then occurs, MVS/ESA must change control-register contents in order to gain access to the home address space so that it can save the status of the step.

To improve the efficiency of accessing the home address space, ESA/370 includes (1) a *home segment-table designation* (HSTD) in a control register and (2) another translation mode, named the *home-space mode*, which is conditioned by bit settings in the PSW. The new PSW that is loaded by the machine when an interruption occurs can specify the home-space mode to provide immediate access to both instructions and operands in the home address space.

SET ADDRESS SPACE CONTROL can set the home-space mode but only in the supervisor state. INSERT ADDRESS SPACE CONTROL can return an indication of the home-space mode.

The instructions that operate on the linkage stack are not allowed in the home-space mode. The linkage stack for a dispatchable unit is in the home address space of that dispatchable unit.

ESA/370 also includes privileged instructions for loading a general register and storing from one through the use of a real address. These instructions allow the control program to process both machine and software control structures that are linked by real addresses without turning DAT off.[20]

Translation modes. Since the ESA/370 virtual-address translation modes have already all been described in this paper, this section is mainly a summary. The modes are controlled by bits 16 and 17 in the PSW. When DAT is on, PSW bits 16 and 17 specify the translation mode as shown in Figure 10. The figure also shows which address spaces contain instructions and operands in each mode.

In ESA/370, unlike in System/370 and 370-XA, instructions are fetched predictably from the primary address space in the secondary-space mode.

In ESA/370 and its predecessors, the central processor always recognizes its own stores into the instruction stream in the primary- or secondary-space mode.[21] Thus, it is possible for a program to modify its next sequential instruction. This recognition is not necessarily done in the two new ESA/370 modes—access-register mode and home-space mode. (It is considered to be poor programming practice to store into the instruction stream. An instruction that causes prefetched instructions to be discarded can be executed to ensure recognition of a store if recognition is necessary.)

Private-space facility. The private-space facility makes the entire 2 gigabytes of specified address spaces available to contain information that is independent of the contents of other address spaces. The facility provides a bit, named the private-space control, in the segment-table designation. When this bit is one in the segment-table designation used by DAT, it makes a translation-lookaside buffer (TLB) entry in which the common-segment bit is one ineligible for use by DAT, even if the segment-table origin field associated with the TLB entry is the same as the one in the segment-table designation. This excludes the common area from the address space specified by the segment-table designation. The private-space control applies regardless of whether the segment-table designation is obtained from a control register or from an ASN-second-table entry by means of access-register translation.

The private-space control, when one, also makes low-address protection (store protection of effective addresses 0–511) and fetch-protection override (not applying fetch protection to effective addresses 0–2047 when effective addresses 0–4095 are fetch protected[22]) not apply to the address space specified

by the segment-table designation, which results in the first 2048 bytes of the address space being as fully usable as the remainder of the space. (An effective address is an address before translation, if any.)

The private-space facility is implemented by some but not all ESA/370 models. When running on a model that does not implement the facility, MVS/ESA excludes the common area from certain address spaces by never setting the common-segment bit to one in any segment-table entry. This exclusion results in a slight performance loss because it requires more frequent creation of TLB entries. Thus, the private-space facility is primarily a performance item, although it does also provide function with respect to bytes 0–2047.

When the private-space control is one in a segment-table designation, the specified address space is called a *private* address space. MVS/ESA calls a private address space a *data space* since it allows only data in the space; MVS/ESA does not assign an ASN to the space, so the space cannot be specified in an entry-table entry. There is no architectural reason why programs cannot be in a private address space.

Unlike the architecture, MVS/ESA strictly uses the dichotomy address space and data space: An address space always contains a common area, and a data space is not an address space.

Concluding remarks

ESA/370 allows increased isolation of both data and programs in separate address spaces. It does so by providing efficient controlled means (access registers and a program-linkage mechanism) of reaching other address spaces when necessary. The benefits are to:

- Increase reliability, meaning the absence of errors that might reduce integrity. By separating data and programs, together and separately, in different address spaces as much as possible, the reliability of the system is improved over what can be accomplished just by means of key-controlled protection. The damage potential of PSW key zero is reduced, and the number of practical domains is increased.
- Increase the storage available in the common and private areas of primary address spaces. By moving data and programs into other address spaces (primarily from the common area), storage is kept available in the primary address spaces for applications that have not been coded to use access registers.
- Increase the total size of the data that can be processed efficiently by a program without control-program intervention. With access registers, data in up to 15 address spaces besides the instruction space can be processed concurrently by means of the complete instruction set, and by changing the contents of the access registers, data in hundreds of other address spaces can be processed without control-program intervention.
- Increase the size of an individual data object that can be processed concurrently by a program. The size of the private area in an address space is 2 gigabytes minus the size of the common area in the space. For example, if the size of the common area is 1 gigabyte, so is the size of the private area. In this case, if a program processes several data objects concurrently (for example, several variables that are arrays), each of those objects must, without access registers, be much smaller than 1 gigabyte in size since all of the objects together (and also the program) must fit within one address space. With access registers, each data object can be in its own private address space and can occupy the entire 2 gigabytes in the space.

ESA/370 allows new tests of the principle[23] that problems expand to fill the storage allowed for their completion.

Acknowledgments

The new facilities of ESA/370 are based on the access-register concept developed by J. R. Butwell, C. A. Scalzi, and R. J. Schmalz. Many elements of the specific architecture were developed by R. I. Baum, T. L. Borden, C. E. Clark, A. G. Ganek, J. Lum, and M. G. Mall. R. M. Smith, J. Thomas, and P. C. Yeh also contributed.

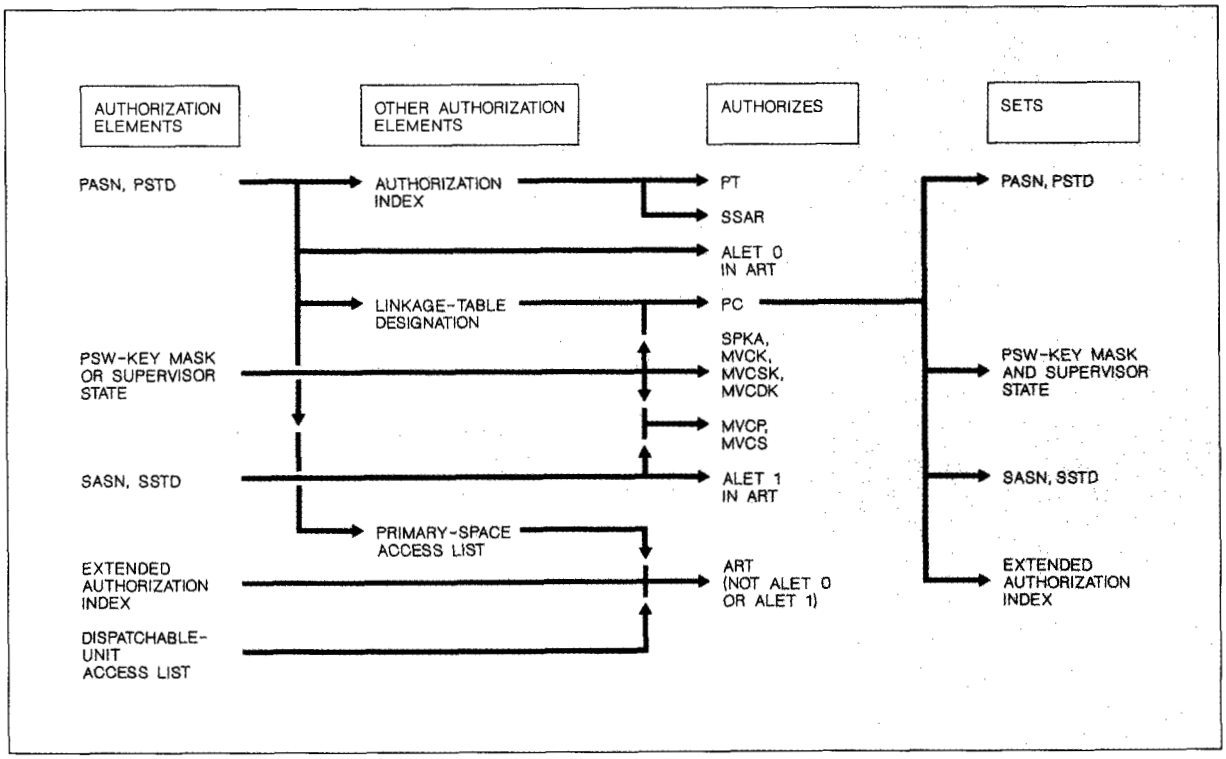
Appendix: Authorization elements

The dual address-space (DAS) facility and ESA/370 contain authorization elements that provide direct and indirect authority to programs. Direct authority allows the execution of specific instructions or the performance of a function that is part of the execution of many instructions. Indirect authority allows changing the state of an authorization element that provides direct authority.

An authorization element is described here in terms of:

- What it authorizes (and also the authorization mechanism if that is unobvious)

Figure 11 Authorization elements



- How its state is set
- What its state is a property of

The authorization elements are listed below. Information pertaining only to ESA/370 is indicated as such. Many of the details of the authorization elements are illustrated in Figure 11. Note that

- DAS introduced the concept of *semiprivileged* instructions that are authorized by certain bits in control registers. The only use of the bits is to provide predictable results (a program interruption) if a program using DAS is executed on a DAS machine under a down-level control program that does not support the use of DAS. The bits are not included in the authorization elements that this appendix describes.
- The symbols PC-ss, PR-ss, and PT-ss are used below to denote the space-switching operations of PROGRAM CALL, PROGRAM RETURN, and PROGRAM TRANSFER, respectively; and SSAR is the mnemonic for SET SECONDARY ASN.

Primary space (PASN and PSTD).

Authorizes:

1. Linkage-table designation
2. Authorization index
3. Primary-space access list (ESA/370 only)
4. ALET 0 (ESA/370 only)

PASN set by:

1. PROGRAM CALL (PC-ss) from an entry-table entry
2. PROGRAM TRANSFER (PT-ss) from a general register
3. PROGRAM RETURN (PR-ss) from a linkage-stack state entry (ESA/370 only)

PSTD set by: Same as PASN, except from the ASN-second-table entry for the specified primary address space

Property of: Entry-table entry

Linkage-table designation.

Authorizes: Which entry-table entries are available to (but not necessarily able to be used by) PROGRAM CALL

Set by: PROGRAM CALL (PC-ss), PROGRAM TRANSFER (PT-ss), and (ESA/370 only) PROGRAM RETURN (PR-ss) from the ASN-second-table entry for the new primary address space

Property of: Primary address space

Authorization index.

Authorizes:

1. PROGRAM TRANSFER (PT-ss) (must select P bit in authority table for specified primary address space that is one)
2. SET SECONDARY ASN (must select S bit in authority table for specified secondary address space that is one)

Set by: PROGRAM CALL (PC-ss), PROGRAM TRANSFER (PT-ss), and (ESA/370 only) PROGRAM RETURN (PR-ss) from the ASN-second-table entry for the new primary address space

Property of: Primary address space

Access list (ESA/370 only).

Authorizes: Which access-list entries are available to (but not necessarily able to be used in) access-register translation

Primary-space access list set by: PROGRAM CALL (PC-ss), PROGRAM TRANSFER (PT-ss), and PROGRAM RETURN (PR-ss) from primary ASN-second-table entry

Property of:

1. Dispatchable-unit access list: dispatchable unit
2. Primary-space access list: primary address space

Supervisor state.

Authorizes:

1. Execution of privileged instructions
2. Omission of the use of the PSW-key mask as an authorization element

Set by:

1. PROGRAM CALL from an entry-table entry
2. PROGRAM TRANSFER from a general register (supervisor state can be set only in the supervisor state)
3. PROGRAM RETURN from a linkage-stack state entry (ESA/370 only)

Property of: Entry-table entry

PSW-key mask.

Authorizes (only necessary in the problem state):

1. SET PSW KEY FROM ADDRESS (mask bit corresponding to specified key must be one)
2. Second access key used by MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, MOVE WITH SOURCE KEY, and MOVE WITH DESTINATION KEY (mask bit corresponding to specified key must be one)
3. PROGRAM CALL (result of ANDing with authorization key mask in entry-table entry must be nonzero)

Set by:

1. PROGRAM CALL by ORing or (ESA/370 only) optionally replacing from an entry-table entry
2. PROGRAM TRANSFER by ANDing from a general register
3. PROGRAM RETURN from a linkage-stack state entry (ESA/370 only)

Property of: Dispatchable unit and entry-table entry or (ESA/370 only), optionally, entry-table entry

PSW key.

Authorizes: Accesses to real storage (must match storage key or be zero)

Set by:

1. SET PSW KEY FROM ADDRESS
2. PROGRAM CALL, optionally, from an entry-table entry (ESA/370 only)
3. PROGRAM RETURN from a linkage-stack state entry (ESA/370 only)

Property of: Dispatchable unit or (ESA/370 only), optionally, entry-table entry

Extended authorization index (ESA/370 only).

Authorizes: Use of a private access-list entry in access-register translation

Set by:

1. PROGRAM CALL, optionally, from an entry-table entry
2. PROGRAM RETURN from a linkage-stack state entry

Property of: Dispatchable unit or, optionally, entry-table entry

SASN and SSTD.

Authorizes: Secondary address space accessed by:

1. MOVE TO PRIMARY and MOVE TO SECONDARY
2. Storage-operand references in the secondary-space mode
3. ALET 1 (ESA/370 only)

Set by:

1. SET SECONDARY ASN
2. PROGRAM CALL with old PASN and PSTD or (ESA/370 only), optionally, with new PASN and PSTD
3. PROGRAM TRANSFER with new PASN and PSTD
4. PROGRAM RETURN (ESA/370 only):
   - SASN from a linkage-stack state entry
   - SSTD from the ASN-second-table entry for the specified secondary address space

Property of: Dispatchable unit or (ESA/370 only), optionally, entry-table entry

Cited references and notes

1. *System/370 Principles of Operation*, GA22-7000, IBM Corporation; available through IBM branch offices.
2. *System/370 Extended Architecture Principles of Operation*, SA22-7085, IBM Corporation; available through IBM branch offices.
3. The definition of *domain* is by Carl E. Landwehr and Brian Tretick (Naval Research Laboratory, Washington, D.C.), John M. Carroll (University of Western Ontario), and Paul Anderson (Naval Space and Warfare Systems Command), and it relates to earlier work by B. W. Lampson.
4. *Enterprise Systems Architecture/370 Principles of Operation*, SA22-7200, IBM Corporation; available through IBM branch offices.
5. R. P. Case and A. Padegs, "Architecture of the IBM System/360," *Communications of the ACM* **21**, No. 1, 73–96 (January 1978).
6. A. Padegs, "System/360 and beyond," *IBM Journal of Research and Development* **25**, No. 5, 377–390 (September 1981).
7. A. Padegs, "System/370 extended architecture: design considerations," *IBM Journal of Research and Development* **27**, No. 3, 198–205 (May 1983).
8. D. Gifford and A. Spector, "Case study: IBM System/360–370 architecture," *Communications of the ACM* **30**, No. 4, 292–307 (April 1987).
9. *American National Dictionary for Information Processing Systems*, available from the American National Standards Institute, 1430 Broadway, New York, NY 10018.

10. Actually, SET ADDRESS SPACE CONTROL and INSERT ADDRESS SPACE CONTROL are authorized by bits in a control register. The only purpose of these bits is to provide predictable results if the SSTD has not been set because a down-level control program is being used.
11. DAS was simulated on the System/370 Models 158, 168, and 3032. This simulation would have been a very poor performer if instructions and data had to be accessed in different address spaces.
12. There are cases where PROGRAM TRANSFER is used by a subsystem to give control to a user exit routine and then PROGRAM CALL is used to return control, in a secure way, to the subsystem.
13. It was said previously that the PSW-key mask of a program is a property of an entry point. This is not strictly true since different PC numbers may specify the same entry point but different entry key masks.
14. An authority table consists of two-bit entries corresponding to different values of the authorization index. One bit (P) authorizes PROGRAM TRANSFER and the other (S) authorizes SET SECONDARY ASN.
15. Performance may also be affected in that, because PROGRAM CALL sets several general registers, the calling program must save and restore these registers around the PROGRAM CALL linkage, if necessary.
16. C. E. Clark, "The facilities and evolution of MVS/ESA," *IBM Systems Journal* **28**, No. 1, 124–150 (1989, this issue).
17. It is, of course, advantageous not to have to change old programs. On the other hand, performance can be improved if instructions that save and restore registers around a PROGRAM CALL linkage are deleted from the old programs.
18. The new significance of the secondary address space is that it can be accessed by means of ALET 1. The called program may also be able to access the primary address space of the calling program if there is an entry for it on the the dispatchable-unit access list.
19. As part of the execution of an instruction, the instruction address in the PSW is updated to address the next sequential instruction. This address is subsequently changed again if the instruction branches.
20. The great majority of the MVS/ESA control program does not reside in virtual-equals-real storage. To turn DAT off, the control program usually must first branch to a special virtual-equals-real part so that the change to DAT is not a branch.
21. A store into the instruction stream is recognized if the same effective address is used for the store and the fetch and the store is not made by the vector facility. Different effective addresses might be used because they can map to the same real address by means of DAT.
22. Fetch-protection override relates to the change of the storage-key block size from 2048 bytes to 4096 bytes that began in

System/370 (although fetch-protection override is not provided in System/370). It is used by evolving control programs that originally chose to fetch-protect the second 2048 bytes of real storage but not the first 2048 bytes (because problem programs fetched pointers from the first 2048 bytes). With these control programs, virtual addresses 0–4095 (a problem program can generally use only virtual addresses) map to real addresses 0–4095, which is why fetch-protection override applies to effective addresses.

23. R. Matick, *Computer Storage Systems and Technology*, John Wiley & Sons, Inc., New York (1977).

Kenneth E. Plambeck *IBM Data Systems Division, P.O. Box 950, Poughkeepsie, New York 12602.* Mr. Plambeck joined IBM in early 1958 and is a senior programmer in the Enterprise Systems Central Architecture department. His initial projects included diagnostic programming for the IBM 709, 7090, and 7950 (a special-purpose extension of the 7030 Stretch computer) computer systems, the indexing and output phases of a FORTRAN compiler for the 7030, and the modification of this compiler to demonstrate the correctness of the number of general registers in System/360. Beginning in 1964, he was in the OS/360 area working on supervisor services, job-scheduler improvements, system-management facilities, checkpoint restart, standards for linkage conventions, volume and file labels, and the development process, and the control of all system control blocks. He began his systems-architecture career in 1972, working first on the FS architecture, then on the VSE architecture, and, beginning in 1979, on extensions to the System/370 architecture. He is the author of the new material in *Enterprise Systems Architecture/370 Principles of Operation.* Mr. Plambeck received a B.E.E. degree from the Georgia Institute of Technology in 1956 and an M.S. degree in electrical engineering from the University of Illinois in 1958.

Reprint Order No. G321-5347.