P. H. Gum

System/370 Extended Architecture: Facilities for Virtual Machines

This paper describes the evolution of facilities for virtual machines on IBM System/370 computers, and presents the elements of a new architectural facility designed for the virtual-machine environment. Assists that have been added to various System/370 models to support the use of virtual machines are summarized, and a general facility for this purpose which was introduced with the System/370 Extended Architecture (370-XA) is described. A new instruction of the 370-XA architecture places the machine in a specific mode in which several special capabilities are enabled. These allow the machine to provide execution in the virtual-machine environment of most of the instructions (including many privileged instructions) and most of the facilities (such as dynamic address translation) of both the System/370 and the 370-XA architectures. The major features of this new facility are individually discussed and summarized.

Introduction

One of the noteworthy and unexpected developments associated with System/360 and then System/370 was the prominence attained in the use of virtual machines. The concept lends itself to interactive use, provides a conceptually simple and complete computing environment for each user, is inherently secure, and allows efficient development of programs from simple ones to complex control programs. A description of how the virtual-machine facilities provided by the IBM Virtual Machine Facility/370 (VM/370) control program are used at the IBM Thomas J. Watson Research Center at Yorktown Heights, New York, may be found in [1]. Considerable value lies in the utility of convenient access to the functions provided in the virtual-machine environment, including the simple and effective file-management and general editing facilities, the high-level languages, the communications possibilities when these systems are linked together in an extensive network [2], and an array of other capabilities.

Virtual machines are the outgrowth of a combination of leading-edge developments of several years ago, including interactive access, dynamic address translation, text editing, and the advent of complex control programs. A history of the development of VM/370 is presented in [3]. A collection of articles covering several facets of VM/370 may be found in [4]. The architecture of System/370 is specified in [5]; a history of the evolution of the machine architecture from System/360 to System/370 is found in [6].

This paper discusses the continuing evolution in sophistication of virtual-machine functions that are incorporated in IBM System/370 computers. The paper is organized into two main parts: Part 1 (System/370 Virtual-Machine Support) provides an introductory overview of the virtual-machine capability on System/370 machines, with emphasis on the development of assists for virtual machines; Part 2 (370-XA Interpretive-Execution Architecture) describes the extensions in the System/370 Extended Architecture (370-XA) provided in support of virtual machines. Extended functions are provided in the areas of multiprocessing, timing, the handling of guest storage and dynamic address translation, the interface with host simulation programs, and the ensuring of integrity between the guest and the host.

A discussion of all of the extensions, not just those for virtual machines, incorporated in 370-XA may be found in

[7]. The 370-XA channel subsystem architecture is described in [8]; both the 370-XA CPU architecture and the channel subsystem architecture are specified in detail in [9].

Originally, all privileged instructions, the maintenance of certain translation tables, and the presentation of interruptions to a virtual machine, called the *guest* system, were handled by a second control program, called the *host* system (an example of which is the VM/370 control program). That is, these functions were provided for the guest through simulation by means of a host program. On the other hand, the execution of problem-program instructions was provided entirely by the machine. Reference [10] provides a more complete introduction to the capabilities of the VM/370 control program. An introduction to the nature of the virtual-machine environment is presented in the first section of Part 1 of this paper (*Virtual-machine concept*).

The number of host instructions executed in simulation routines, and the frequency with which the routines are used, can result in the consumption of significant amounts of CPU time. This has encouraged the design of machine functions referred to as *assists* to be added to the CPU to perform some of the functions that were previously simulated. Assists reduce the number of instructions that must be executed for these overhead activities; the effect is to shift the processing resource back to the execution of guest instructions, often dramatically improving both the number of virtual machines supported and the responsiveness to them. The variables affecting performance in a virtual machine are discussed in the second section of Part 1 (*Performance variables*).

Over time, quite a number of assists have been created to improve the operation of a VM/370 system [11]. Most of these provide for the execution of guest instructions which otherwise would have been simulated. However, some assists provide improvement by supplanting frequently executed portions of the host control program [12]. Not all assists have been made available on all systems; sometimes only a subset of an assist is provided on a particular model. A summary of the development of assists for VM/370 is given in the third section of Part 1 of this paper (*Evolution of assists*).

A primary objective of the 370-XA interpretive-execution architecture is the realization of benefits from a comprehensive extension of the capabilities previously available only through assists. The capability provided is thus a continuation of the evolution of assists. Most previous assists are subsumed in some form under a basic added function of the machine, referred to as an *interpretive-execution capability*. This is provided as a mode in the machine which causes it to recognize the special handling of instructions, facilities, and events that is necessary in a virtual-machine environment. The 370-XA interpretive-execution capability is invoked by means of an instruction, the operand of which is a control block in storage. This block describes the "machine" to be executed, including the values of some of the registers of the virtual CPU and the storage available for use by the guest. The capability is provided for the execution of virtual machines in which either the System/370 or the 370-XA architecture is used by the virtual machine.

Part I: System/370 virtual-machine support

• *Virtual-machine concept*

Suppose we envision an operating system, the CPU on which it is running, the main storage it is using, and the existing assemblage of I/O equipment as stopped at an instant in time. Imagine moving the control and application programs intact—main storage contents and the contents of the CPU registers—to another computer-system environment, along with copies of requisite files. Then, let execution resume at the next sequential instruction. The programs have been moved from execution in a *native* environment to execution in a *virtual-machine* environment. The new environment is in essence or in effect (i.e., virtually) the same as the old environment: ideally the system that is moved cannot detect a difference, though in fact the environments are different. The system that is moved is referred to as the guest system. The environment to which it is moved is provided by another control program called the host system. While the change is in practice not made so abruptly, this scenario conceptually focuses on a major goal of a virtual machine of making real circumstances transparent to a guest. The objective in fashioning the environment in which the guest programs are placed is to cause the same changes to occur for things to which the programs have access—registers, files, and storage—as would occur in any other environment. In VM/370, this may be regarded as being achieved by techniques which provide equivalents or imitations of the places where these things originally resided. This often involves a translation of a guest "address"—a storage address, device address, or cylinder address—into a host address at which VM/370 (the host) is maintaining the information for the guest (a means is always provided to cause control to be given to the host when an unaltered value is about to be used by the guest). For example, usually a complete replacement is used for a guest channel program, accommodating at once substitutions for storage addresses, device address, and device characteristics. There are, on the other hand, cases where substitution is typically not used; storage keys and timing facilities, for example, are used directly by the guest, though the use is shared somewhat with the host.

The representation of guest main storage by a portion of a host address space typifies the utility of substitution, in this case using virtual storage as a replacement for real storage. This tactic relies on the characteristic that most programs

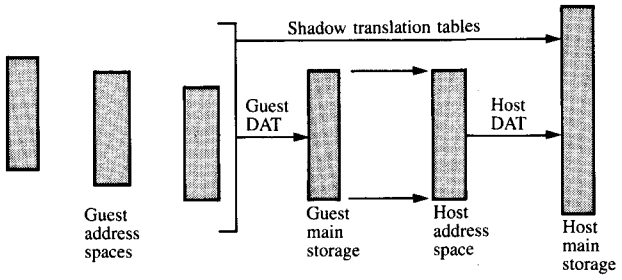

Figure 1 VM/370 address translation.

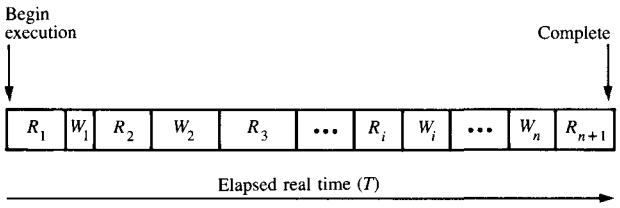

Figure 2 Run-time intervals. R_i = the ith instruction-execution interval, $i = 1, 2, \cdots, n$. W_i = the ith idle interval (waiting, overhead, handling other work, etc.), $i = 1, 2, \cdots, n$.

(with few exceptions that are dealt with in other ways) will in fact execute correctly without change in either a real-address or a virtual-address space. The machine provides a mechanism for one level of translation—from a virtual address to a real storage address. The use of *dynamic address translation* (DAT) by the guest, which implies translation from a guest virtual address to a host virtual address in the host address space containing guest main storage, is not directly provided by the machine. The basic machine mechanism uses real translation-table-entry addresses, and thus cannot use guest tables which reside in a host virtual-address space. The technique employed by VM/370 is to use *shadow translation tables*. These are tables which are usable by the machine in the usual way, but their contents have been supplied by VM/370 such that a translation of a guest virtual address produces a corresponding real (host) storage address. A VM/370 program (or the page-validation assist) accomplishes this by treating guest translation-table-entry addresses as host virtual addresses. In this way, a guest virtual address is first converted to a guest "real" address; that address is equivalent to a host virtual address. The host virtual address is then converted to a host real-storage address, which is then placed in the shadow-page-table entry. This procedure takes a minimum of eight storage references to obtain the host real address corresponding to a guest virtual address. Additional references are made to VM/370 control blocks to obtain the origins of the host-managed translation tables, and to fill in the shadow-page-table entry. Figure 1 illustrates the address spaces and translations involved.

The host program maintains control over the use of the real machine registers—*program-status word* (PSW), control registers, timing—by placing the machine in problem state when the guest is given control. Thus, the privileged operations used by the guest cause an exception to be recognized, returning control to the host program before the contents of a register are changed and providing the host overall control of the real configuration. Registers which are changeable by problem-program instructions—the general registers and the floating-point registers—are given over fully to the guest. The design of many assists relies on the recognition of the exception for privileged operations. The assist is given control after the exception is recognized but before the host control program is given control. The assist first examines the circumstances under which it has received control and then either completes the original function on behalf of the guest or allows control to revert to the host, normally for the original exception.

• *Performance variables*

This section identifies some of the aspects of execution that contribute to a difference in the performance of a program executed in a virtual machine and the same program executed natively. Figure 2 illustrates the salient features of execution in a native environment: typically, run time consists of a sequence of alternating intervals of execution and waiting. The durations of both kinds of intervals are irregular. A typical way of measuring system performance is to present a workload to the system and measure the real time taken to complete the work. The CPU spends some time executing the instructions comprising that workload, the R intervals, and some time idling or executing other work, the W intervals. The sum of both kinds of intervals taken together is a measure of performance.

In a virtual-machine environment, both the rate at which instructions are executed and the frequency and duration of periods of execution and suspension change. Let the time taken natively to execute a workload be represented by T_n, and let the time taken in a virtual machine for the same workload be represented by T_v. A commonly used measure of execution performance in a virtual machine is *relative batch throughput* (*RBT*), defined as the ratio of the native execution time to the virtual-machine execution time, or

$$RBT = T_n/T_v.$$

A general objective is to achieve a value for this ratio approaching 1, which would indicate a virtual-machine execution time equal to native execution time. The effectiveness of an assist can be expressed in terms of the improvement in *RBT* that is observed when the assist is utilized.

Other useful measures of the effectiveness of an assist are its reduction of supervisor-state busy time or its reduction of

CPU busy time. That is, the effectiveness of an assist can also be evaluated by its ability to reduce the number of machine cycles required to perform the work, since in a time-sharing system, efficient use can be made of additional available CPU time to execute other work.

Figure 3 illustrates the factors that elongate the execution time of a given workload in a virtual machine. The additional factors are indicated by the symbols *E*, *X*, *S*, *T*, *I*, and *M*. They characterize the intervals according to criteria pertinent to the virtual-machine environment. They are shown as additional types of, or effects on, intervals and represent additional work that must be accomplished in a virtual-machine environment. The additional factors may be summarized as follows:

- *E* The added overhead to dispatch a guest, including setting up the timing facilities, program-status word, and control registers.
- *X* The added overhead to store away guest status and reestablish the host environment when execution of the guest is discontinued.
- *S* Simulation of guest instructions by a host program.
- *T* Guest wait-state handling, usually involving establishment and then deletion of real-time-interval monitoring of the guest wait period.
- *I* Interruption handling, usually involving handling of the interruption twice—once by the host and once by the guest.
- *M* Even instructions which are executed by the machine are subject to apparent elongation of execution time. There are two principal contributors: 1) for some instructions, tables must be referenced in the virtual-machine environment that are not used natively, and 2) such things as address translation, which occurs for multiple layers of addressing in the virtual-machine environment, have the statistical effect of making all instructions appear to take longer on the average to execute.

A fundamental purpose of assists and the 370-XA interpretive-execution capability is to diminish the effect of one or more of these factors.

• *Evolution of assists*

The term *assist* is applied to a function which is to be distinguished from the basic architecture. The principal reason for this distinction is to call attention to a function normally not considered usable outside the environment of a specific control program. Often the function has a dependency on a control-block structure that is normally established only by a particular control program. That is, there is an implied reliance on the structure being used by the machine function in the same way the control program uses the structure.

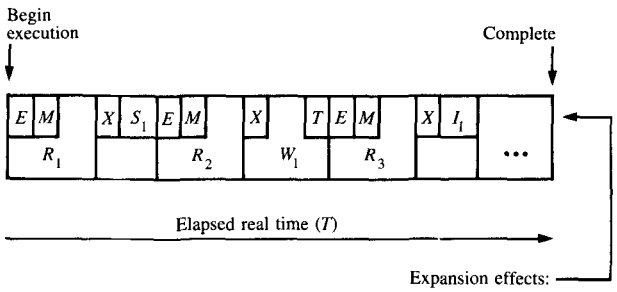

Figure 3 Virtual run-time intervals.

Many assists also have the characteristic that correct execution of the control program does not depend on their use, their value lying instead largely in their ability to improve performance. This characteristic has allowed the VM/370 control program to be run on a variety of models, some with different assist capabilities and some with different levels of the same assist.

The following sections briefly review the assists provided for use by VM/370. Different assists attack different factors contributing to the expanded time for execution in a virtual machine. Another comprehensive discussion of assists for VM/370 may be found in [13].

Virtual-machine assist

The *virtual-machine assist*, commonly referred to as VMA [11], was developed almost a decade ago. As much as anything, the development of the assist was stimulated by the need to maintain virtual-machine performance as guests began to make use of DAT. The assist consists of 13 functions, 12 of which accomplish execution for the guest of one problem-program and 11 privileged instructions which otherwise would be simulated by the host program. The 13th function takes over from the host program certain aspects of the management of tables used by the host program in support of the guest. The functions for which assists are provided by VMA are shown in Table 1.

This collection of assists dramatically improved virtual-machine performance for some types of guests. Of the factors described in the preceding section "*Performance variables*," for the assisted instructions the assists nearly eliminated factors *E*, *X*, and *S* from the expanded virtual-execution intervals. Improvements in *RBT* from a value of 0.35 to a value of 0.70 or higher were not uncommon for certain types of guest control programs [13]. VMA demonstrated value not only in assisting specific guest instructions, but also in subsuming frequently occurring host support operations, exemplified by the page-validation function. This success set the stage for extending the assist approach to fit more and different kinds of guests, and for taking advantage of the

Table 1 Virtual-machine assist functions.

Assist collection	*Number of functions*
Virtual-machine assist (VMA)[1]	13
Extended control program support:	
Control program assist	22
Expanded virtual-machine assist	12
Virtual interval-timer assist	1
Shadow-table-bypass assist	8
Preferred-machine assist	22
Dual-address-space assist	20
Extended-storage-key assist	3
	Total: 101

The following functions caused changes to many of the above assists:

- Common-segment bit
- Segment protection
- Low-address protection
- 26-bit real addressing

Provision for use in a virtual machine was included in the base definition of the assists for MVS (14 functions).

[1]The assists for the following functions comprise the virtual-machine assist:

INSERT PSW KEY (IPK)
INSERT STORAGE KEY (ISK)
LOAD PSW (LPSW)
LOAD REAL ADDRESS (LRA)
RESET REFERENCE BIT (RRB)
SUPERVISOR CALL (SVC)
SET STORAGE KEY (SSK)
SET SYSTEM MASK (SSM)
STORE CONTROL (STCTL)
STORE AND AND SYSTEM MASK (STNSM)
STORE THEN OR SYSTEM MASK (STOSM)
SET PSW KEY FROM ADDRESS (SPKA)
Shadow-page-table validation

characteristics of different classes of machines. This initial effort demonstrated the enormous potential in the approach of using assists.

In testimony to its usefulness, VMA is a rare instance of a package of functions designed specifically for a particular control program that was eventually used by another control program, the specialized Airlines Control Program (ACP) [14].

Extended Control Program Support

The next major collection of assist functions were developed under the name Extended Control Program Support:VM/370 (ECPS:VM/370). Thirty-five distinguishable functions are provided in this collection. They represent a response to identified opportunities, a focus on the kind of guest that is of predominant interest to users of intermediate-scale System/370 machines, and an effective utilization of the design characteristics of this class of machines. In addition to incorporating VMA, ECPS:VM/370 assists in the maintenance of the interval timer, assists more guest instructions, assists in more circumstances some of the same instructions originally assisted by VMA, assists the handling of I/O, and provides assists which are replacements for 22 sections of the host program. A more complete description of these functions appears in [12].

Specialized support

With the efficacy of a wide range of assists established, attention then focused on special situations. Examples are the so-called *virtual-equals-real* (V = R), or *preferred*, virtual machine, and the use of shared segments under the Conversational Monitor System (CMS) operating system. A common mode of operation consists of supporting many on-line terminals by means of individually dedicated virtual machines, each normally under control of a CMS guest operating system. In addition, a single virtual machine, often heavily used, through which batch work is scheduled, is usually provided. The batch system is usually run as a V = R virtual machine.

The V = R guest is executed with the host address space representing guest main storage mapped one-to-one onto real main storage (from which the V = R designation comes), except for usually one or a few pages. The primary benefit is that most channel programs for a V = R guest can be executed as is, eliminating the overhead of a host program having to construct copies with valid real (host) addresses. The usually tolerable exposure is that an errant guest channel program might read or write real storage outside the range assigned to the guest. The CPU, however, continues to process guest instruction and operand addresses through translation tables whose validity is controlled by the host

program. This includes the use of shadow translation tables when the guest enters DAT-on mode. Two classes of assists have been provided for this environment.

Shadow-table-bypass assist: If there is sufficient trust in the reliability of the guest, use of the shadow translation tables can be dispensed with, relying instead mostly on the translation tables provided by the guest. The guest tables are not used wholly as is because the guest is not normally given control of real page frame zero, which previously was "hidden" by a suitable adjustment of the shadow translation tables. The *shadow-table-bypass assist*, described in [11], accommodates the handling of special page frames, yet generally allows use of guest page-translation tables as is, with a consequent benefit to performance. Experience has shown that generally satisfactory operation is achieved with reliable guests.

Preferred-machine assist: More recently, this idea of relying on the well-behaved characteristic of some guests has led to the *preferred-machine assist* (PMA). With this assist, not only are host translation tables not used, but the guest is allowed to run in the real supervisory state and is given access to real page frame zero. Most uses of privileged instructions are thus executed for the guest essentially as native instructions. This includes the execution of most I/O instructions, such as START I/O FAST RELEASE (SIOF), for devices attached to real channels considered "dedicated" to the guest. Under PMA, only minimal checks are imposed on the use of privileged instructions by the guest; most normal operations by the guest, privileged or not, involve no intervention by the host program. Violation of a check, or the establishment of the pending state of an I/O interruption not intended for the guest, cause control to revert automatically to the host.

Segment protection: Another important special situation arises from the extensive use of several concurrently on-line virtual machines, each of which is under control of a CMS. Advantage is taken of this to reduce paging traffic by sharing segments of commonly used programs. Previously, special tests were made, by programmed means at a performance cost, to detect improper changes to areas that could otherwise be shared. A performance improvement is achieved by incorporating a protection mechanism at the segment level that enables the host program to prevent storing into certain segments of storage, eliminating the need for the special testing. Segment protection is not considered to be an assist but rather an extension of the base architecture.

Effects of functional enhancements

Over time, enhancements were also added to the System/370 architecture for purposes independent of VM/370. Invariably, however, such developments must be considered in the VM/370 context. Because of the ease with which the environment of a virtual machine is controlled and examined, it has become the primary vehicle for the development of control programs, with the consequence that almost all new architectural enhancements are immediately sought for use in the appropriate virtual-machine environment. Depending on the facility, VM/370 either a) ignores the facility because it is unaffected by its presence, b) requires modifications of various existing assists, or c) does not allow guests to use the facility. Some examples of the variety of ways in which new functions are accommodated in the virtual-machine environment are the following. For the *assists for MVS*, provision for operating in a virtual machine is incorporated in the native definition of the facility (MVS denotes the control program for Multiple Virtual Storages). In the case of the *dual-address-space* (DAS) facility, new assists were devised. In the case of the introduction of 4K storage-protection keys, old assists were modified to the unusual extent of providing function not originally available natively for the assisted instruction; in addition, new assists were added for new instructions. Extended addressing, with addresses of either 25 or 26 bits, depending on the model, causes changes to several assists. However, extended addressing is not made available for use by the guest (except under PMA).

Summary of the development of assists

More than 100 individual assist functions have been defined for use with virtual machines. This number does not include 12 of the instructions of the assists for MVS whose basic design incorporates provision for operating in both the native and a virtual-machine environment. Among the assisted instructions are nine which are multiply assisted, some in as many as three different ways. That is, depending on the natures of the particular virtual machine and the instruction, one of the two or three assists available for that instruction is invoked. There are three variations of page-exception handling, in addition to the handling provided by one of the assists for MVS. There are almost three dozen individual changes to existing assists to accommodate subsequent developments, such as DAS, 4K-byte key blocks, 26-bit real addressing, and protection and common bits in DAT segment-table entries.

At times, this diversity is a source of confusion. It sometimes costs extra machine resources (microcode space and performance) because of the lack of a sharing of common subfunctions. Each machine model usually offers a distinct collection of assist functions. The effectiveness of the particular collection of assists on any one machine usually depends on the particular guest control program, and sometimes even on the particular release of that control program. Still, these assists enable a variety of guests to be run on a variety of IBM System/370 models, generally at quite acceptable performance levels.

The achievements of the assists for VM/370, especially the 22 functions added to support the host control program,

were instrumental in encouraging efforts to define assists for other control programs, including Virtual Storage/1 (VS/1) and MVS, some of which have been carried into 370-XA as well.

This brief description of assists associated with VM/370 has not touched on other types of assists, such as, for example, the assist provided on some models for A Programming Language (APL). Nor is the Disk Operating System (DOS) assist described, which is a functional forerunner of the facility described in the next part of this paper. The DOS assist allows a DOS system that does not use DAT to be executed as a guest of the MVS control program in much the same way a problem program is handled by MVS. A list of such additions to certain models and references to more complete documentation are contained in Appendix D of [5].

Part 2: 370-XA interpretive-execution architecture

A principal objective in the development of the 370-XA interpretive-execution architecture was to provide comprehensive support of the virtual-machine environment. This included providing, in the virtual-machine environment, the facilities of both the new and precursor architectures. Interpretive execution of System/370 provides a way of running the machine in the new extended-architecture mode while continuing to make major use of programs using the previous System/370 architecture, thus adding a degree of flexibility in migrating to the new architecture and mode. Interpretive execution of 370-XA aids in the development, checkout, and use of new or changed programs which use the new facilities of the extended architecture.

A primary goal was to make the facilities of either architecture usable at reasonable performance by a variety of users employing the facilities in a variety of ways on a range of machines. Providing full handling for more instructions and facilities and providing the same complement of functions uniformly on all machines avoids uncertainties regarding just which aspects of an instruction or facility are most usefully assisted. Providing comprehensive capabilities is a natural extension of the growth over time of the number of assists and the completeness with which they were providing execution in the virtual-machine environment. This tends to make virtual-machine support less specialized to particular guest systems, and it also makes machines more interchangeable.

Additional objectives were to make the interpretive-execution capability, or at least parts of it, usable by more than one host control program, thus broadening the usefulness of the facility, and to provide a base into which it would be simpler to incorporate future extensions to the architecture. Since the design and development of new inter-component interfaces within an established control program are always difficult and time-consuming, an additional objective was to minimize the number of new control-program interfaces implied in supporting the architecture.

The following sections describe the significant functional aspects of the 370-XA interpretive-execution architecture. Emphasis is placed on those aspects of the architecture that are a departure from capabilities previously provided. The major features are the following:

- An instruction is provided that establishes a mode in the machine in which instructions and facilities are interpreted for the virtual-machine environment. This is called *interpretive-execution mode.*
- Interpretive execution of two architectures is provided, either the System/370 architecture or 370-XA.
- With the exception of the I/O instructions, most privileged instructions are completely executed in the virtual environment. In most cases, an option is also provided for individual instructions which causes control to be returned to the host when the instruction is encountered in the guest. In addition, most program interruptions are handled entirely within the guest.
- Guest main storage is represented either by the corresponding real storage or by a portion of a host address space, variable in amount in both cases and beginning at an offset when a host address space is used. Shadow translation tables are not normally used. Prefixing is provided as required for operation of a guest multiprocessing system. As appropriate, 24-bit, 26-bit, and 31-bit addressing are provided to the guest. Depending on the model, under certain circumstances address translation is accomplished at native performance. Guest programs are prevented from accessing storage outside storage assigned for use by the guest.
- A full complement of guest timing facilities is provided. Interval timing is provided for System/370-mode guests (even though there is no interval timer natively in the 370-XA mode). The value of the guest TOD clock can be set separately from that of the host TOD clock. Host timing is unaffected by the commencement of timing for a guest.
- Most facilities of the architecture, including, for example, DAS and *program event recording* (PER), are provided for the guest, generally without affecting the use of the same facilities by the host.
- All forms of protection provided in the architecture are provided on behalf of the guest. In addition, in pageable-storage mode, host page protection is in effect for guest store accesses to guest main storage.
- Information is provided on exit from interpretive-execution mode concerning the reason for the exit, to improve the efficiency with which the subsequent host program

handles the condition which caused the exit.

- Capabilities are provided for both host and guest multiprocessing. Means are provided for the host to provide asynchronous indications of events of interest to the guest. Interlocks are provided to control access to shared resources among the guest CPUs, and between the guest and host programs.
- In a preferred-storage mode in which guest channel programs are used as is, a capability is provided whereby the machine monitors the addresses used by guest channel programs: an optional control may be set, on a per-subchannel basis, to prevent guest channel programs from accessing storage outside the storage limits assigned for use by the guest.

Most of these topics are covered in more detail in the following sections.

- *START INTERPRETIVE EXECUTION (SIE) instruction*

Interpretive-execution mode, that is, the mode in which the instructions of a virtual machine are directly executed by the machine, is entered by means of the privileged START INTERPRETIVE EXECUTION (SIE) instruction. The operand of this instruction, called a *state description*, defines the environment of the guest system. The environmental information falls into four general categories. One category consists of the contents of various registers of the guest CPU. The second defines how the guest is to fit into the host system, mainly specifying how much and what kind (real or virtual) of host storage is to be used for guest main storage. Also in this category are designations of additional satellite control tables. The third category includes controls over the use of various facilities and instructions of the architecture by the guest. The fourth category consists of the specialized information developed on exit from interpretive-execution mode for use by the subsequent host program. The contents of the state description are summarized in Table 2.

Table 2 Contents of the state description.

- Architecture mode selection:
  - System/370 or 370-XA
- Storage definition:
  - preferred or pageable mode
  - prefix
  - offset (origin within host address space)
  - extent (amount of guest main storage)
- Program-status word (PSW)
- General registers 14 and 15
- Control registers
- Timing:
  - residue (interval-timer accumulator)
  - CPU timer
  - clock comparator
  - TOD epoch offset
  - interval-timing-enablement control
  - interval timer pending interruption indicator
- Intervention controls (which can be set asynchronously):
  - pending I/O interruption
  - pending external interruption
  - pending stop (operator control interpretation)
- Instruction and facility interception controls
- Interception information:
  - bytes 1 and 2 of the instruction
  - operand address information
  - interception reasons:
    - instruction
    - instruction and program interruption
    - I/O intervention
    - validity
    - operation exception
    - program interruption
    - external intervention
    - external interruption
    - wait state
    - stop
  - interception status indicators:
    - I-fetch PER applies
    - interception applies to target of EXECUTE instruction
- Satellite table origins
- Parameters of a guest interruption:
  - program interruption
  - external interruption (mandatory interception)

- *Interception*

Exit from interpretive-execution mode occurs in two general ways. One is by *interruption*, with control going to the host interruption handlers. The SIE instruction is designed according to the criteria of interruptible instructions for this purpose; the instruction address that is recorded in the host interruption old program-status word designates the location of the SIE instruction.

The second method of exit is to return control to the host program at the instruction following SIE, which may be considered to have been completed in this case. Exit in this form is normally induced by a condition encountered in the guest which requires treatment by a host program. This process of leaving interpretive-execution mode is termed *interception*. The three main functions of interception are the following:

a. Storing into the state description the status of the guest that will be needed for resuming the guest.
b. Storing information that will be convenient to the host program that deals with the particular reason for the interception.
c. Restoring the host program.

The general flow of entry to and exit from interpretive-execution mode is shown in Fig. 4.

Among the causes for interception are the following:

- Instructions which are not executed (for which "execution" is usually provided through simulation by a host program). Interception is mandatory.
- An exception condition for which control is always given to the host. Interception is mandatory.
- An instruction or interruption whose execution depends on whether a control bit is set; i.e., interception is condi-

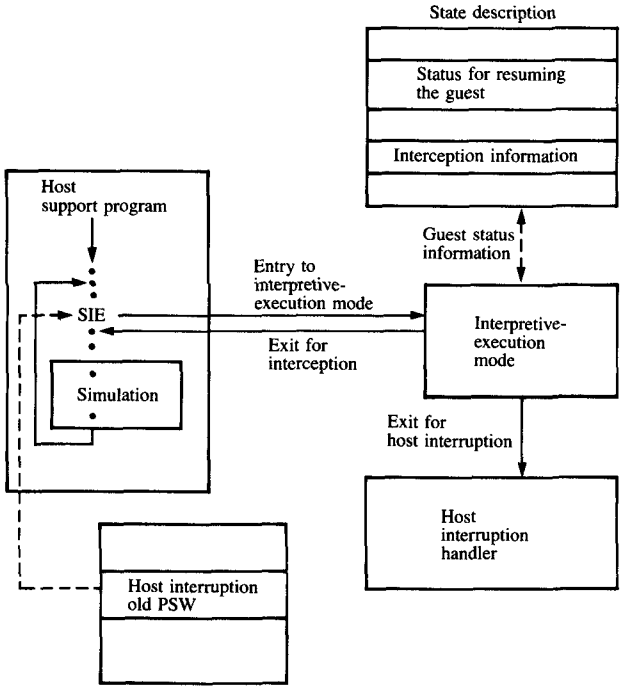

Figure 4 Interpretive-execution entry and exit.

tional. Depending on the instruction, execution may be either suppressed or completed at interception. In some cases, interception is recognized only for certain aspects of execution, normally selected by a mask, or only when certain results are obtained during execution.

- An externally set intervention condition is detected (these conditions are more fully discussed in a subsequent section on intervention requests).
- A special case is recognized, such as recognizing that the guest has entered the wait state.

The efficiency of the process of simulating an instruction is improved by providing several pieces of information about the instruction, including the following:

- Whether the instruction is the target of an EXECUTE instruction.
- Whether an I-fetch PER event is applicable.
- At least the first two bytes of the instruction.
- Either the complete instruction, the effective-operand-address value or values, or the values designating the general registers containing operand information. In some cases, such as in the case of the DIAGNOSE instruction, special handling is provided.

A summary of the handling of guest instructions is given in Table 3.

- *Timing*

The machine maintains both a host set and a guest set of timing facilities while in interpretive-execution mode. The timing facilities for an individual guest are, however, maintained only while the machine is in interpretive-execution mode for that guest. Separate time-of-day (TOD) clock values are provided for the host and the guest, and each control program can use, respectively, a CPU timer and clock comparator. An interruption, when due, is appropriately generated for either the host or the guest, depending on whether the request arises from a guest or a host timing-facility condition. A constant kept in the state description, which represents the difference in the epochs of the guest and the host, is used to generate a TOD-clock value for the guest that is independent of the value of the host TOD clock.

The interval timer, in location 80 in storage, is optionally maintained for a System/370 guest. The interval-timer stepping is sufficiently infrequent that it is possible to enter interpretive-execution mode, do useful work, and then exit from interpretive-execution mode without consuming enough elapsed time to cause a decrementing of the timer. Work could appear to be accomplished "for free" since no time was charged. This is avoided by the use of an additional time-accumulation mechanism that has an accuracy comparable to that of the TOD clock. When sufficient time accumulates, it is posted to the interval timer as an additional decrement. The precision with which the interval timer is maintained is, however, model dependent; that is, decrementing may occur only in multiples of the minimum interval of the timer. Although the time between updates varies by model, it is constrained to be roughly related to processor performance.

- *I/O support aids*

The following items constitute the facilities provided in support of the handling of guest I/O:

- Except for the TEST CHANNEL (TCH) instruction for System/370-mode guests, guest I/O instructions cause interception. The information provided at interception contributes to the efficient handling of the functions by the host.
- A bit pattern with a correspondence to System/370 channels is used by the TCH instruction for recognizing interception or for completing execution by setting a condition code.
- Intervention-request bits, asynchronously settable by other host CPUs, are interrogated regularly to normally cause interception only when the guest is enabled (see the subsequent section on intervention requests).
- Three functions are incorporated in the new 370-XA channel subsystem specifically for support of virtual machines:

Table 3 Guest instruction handling.

	370	*370-XA*
Privileged instructions:		
Mandatory interception:		
I/O related	11	13
Others[1]	8	9
Conditional interception:		
I/O related	1	0
Others[2]	22	19
Always executed[3]	12	13
Problem-program instructions:		
Conditional interception[2]	5	5
Always executed (all others)	—	—

Notes:

[1]Instructions that are not executed (other than I/O):

DIAGNOSE
SET CLOCK (SCK)
SIGNAL PROCESSOR (SIGP)
SET PREFIX (SPX)
START INTERPRETIVE EXECUTION (SIE)**
STORE CPU ADDRESS (STAP)
STORE CPU ID (STIDP)
STORE PREFIX (STPX)
TEST BLOCK (TB)

[2]Conditionally executed instructions:

a. Privileged instructions:

INSERT STORAGE KEY (ISK)*
INSERT STORAGE KEY EXTENDED (ISKE)
INVALIDATE PAGE TABLE ENTRY (IPTE)
LOAD CONTROL (LCTL)
LOAD ADDRESS SPACE PARAMETERS (LASP)
LOAD PSW (LPSW)
PROGRAM CALL (PC)
PROGRAM TRANSFER (PT)
PURGE TLB (PTLB)
RESET REFERENCE BIT (RRB)*
RESET REFERENCE BIT EXTENDED (RRBE)
SET CLOCK COMPARATOR (SCKC)
SET CPU TIMER (SPT)
SET STORAGE KEY (SSK)*
SET STORAGE KEY EXTENDED (SSKE)
SET SYSTEM MASK (SSM)
STORE CLOCK COMPARATOR (STCKC)
STORE CONTROL (STCTL)
STORE CPU TIMER (STPT)
STORE THEN AND SYSTEM MASK (STNSM)
STORE THEN OR SYSTEM MASK (STOSM)
TEST PROTECTION (TPROT)

*System/370 only

b. Problem-program instructions:

COMPARE DOUBLE AND SWAP (CDS)
COMPARE AND SWAP (CS)
STORE CLOCK (STCK)
TEST AND SET (TS)
SUPERVISOR CALL (SVC)

[3]Privileged instructions that are always executed:

EXTRACT PRIMARY ASN (EPAR)
EXTRACT SECONDARY ASN (ESAR)
INSERT ADDRESS SPACE CONTROL (IAC)
INSERT PSW KEY (IPK)
INSERT VIRTUAL STORAGE KEY (IVSK)
LOAD REAL ADDRESS (LRA)
MOVE TO PRIMARY (MVCP)
MOVE TO SECONDARY (MVCS)
MOVE WITH KEY (MVCK)
SET ADDRESS SPACE CONTROL (SAC)
SET PSW KEY FROM ADDRESS (SPKA)
SET SECONDARY ASN (SSAR)
TRACE (TRACE)**

**370-XA only

- A checking mode can be enabled on an individual subchannel basis that prevents execution of a *channel-command word* (CCW) that refers to storage beyond a specified limit. Normally this is the limit of storage assigned to a preferred guest.
- For supervisory uses by the host, primarily to retrieve status for some control-unit malfunctions, a control allowing an override of the limit check in an individual instance is provided.
- To permit the condition code to be set correctly and in a timely fashion for a START I/O (SIO) instruction, an interruption can be requested from designated subchannels when an I/O operation is initiated.

Additionally there is the preferred mode, which eliminates the need for the host control program to construct copies of guest channel programs. In pageable mode, copies of guest channel programs are constructed in the host in which guest

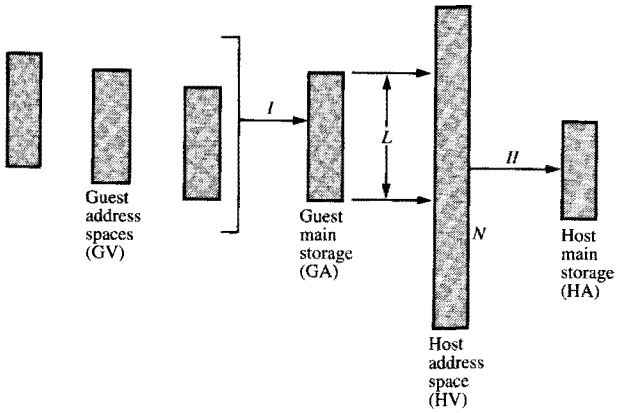

Figure 5 Pageable-mode address translation.

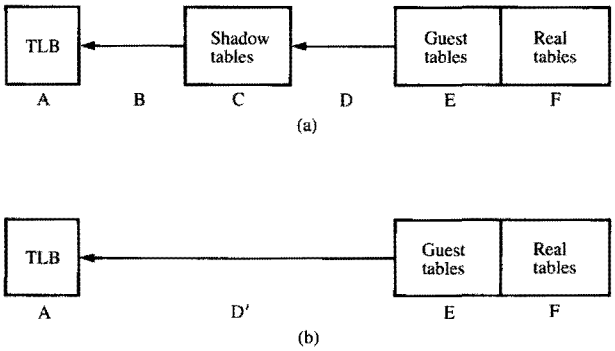

Figure 6 Translation-mechanism differences: (a) System/370 (shadow-table maintenance); (b) 370-XA (SIE).

addresses are replaced with real storage addresses, also simultaneously verifying that the designated storage has been assigned to the guest. The absence of overhead in the preferred mode to perform the conversion improves performance, and the use of dynamically modified channel programs is allowed.

• *Storage*

One of the most distinctive aspects of virtual machines is the ways in which guest "main storage" is represented. Two distinctly different techniques are used. One technique represents guest main storage by a portion of a host address space. A constant is added to each guest absolute address to form a host virtual address which is then translated in turn to a host main (real) storage address. Guest real page frame zero must be "fixed"; i.e., a host page-translation exception on access to this particular frame of guest storage is treated as an error. This is called the *pageable-storage mode*. With the second technique, guest absolute addresses are considered to be the corresponding host absolute addresses. The machine makes no use of host DAT with this method, referred to as the *preferred-storage mode*. With both techniques, the use of guest DAT is allowed. Further, guest prefixing is always applied.

Figure 5 illustrates the address-translation mechanism provided by the SIE instruction. In the most general case, a guest virtual address (GV), when pageable mode is specified, is translated (*I*) by use of guest translation tables residing in guest main storage. After the application of guest prefixing, a guest absolute (GA) main-storage address is obtained. That address is verified to lie within the allowed extent, shown as *L*, and is then converted to a host virtual (HV) address by the addition of an offset, shown as *N*. The host virtual address is translated, indicated by *II*, by using host translation tables, into a host absolute (HA) address. A guest-type *translation lookaside buffer* (TLB) entry is made, consisting of the guest virtual address and the corresponding host absolute address.

Figure 6 illustrates the differences in the translation of guest virtual addresses under VM/370 and under 370-XA. The symbols (A), (B), · · · refer to parts of the figure. Under VM/370, a "miss" in the translation lookaside buffer (A) causes a reference to be made to the current translation tables (C). With some probability, the address is translated by using these tables, with the result returned (B) to (A). If a "page fault" is recognized, reference is then made to guest translation tables (E), such references involving additional subsequent references to the host tables (F) which provide translation of the addresses of the host space containing guest main storage. If the translation is successful, a real address from the host table (F) is placed (D) in the current tables (C), and the whole translation process is retried. When the translation is not successful because of the contents of the (F) tables, a "page fault" is recognized in the host. When translation is not successful because of the (E) tables, a "page fault" is simulated for the guest. When VMA is installed, the accesses to tables (E) and (F), and the updating (D) of the page tables of (C) are handled by the page-validation function.

The performance of this mechanism is highly dependent on the probability of a successful translation on the first access to tables (C). Performance is also a function of the efficiency of use of tables (E) and (F), and the probability of obtaining a successful translation from them.

The contents of tables (C) are sensitive to the characteristics of the guest. Changes to the contents of guest tables (E) must be reflected in the current tables (C). Such guest operations as LOAD CONTROL (LCTL), PURGE TLB (PTLB), and INVALIDATE PAGE TABLE ENTRY (IPTE) must also result in changes to the current tables, which sometimes cause deletion and reconstruction of the

tables. To reduce the performance impact of handling guest IPTE instructions, a special additional table structure must be maintained if shadow-table entries are also to be selectively invalidated. Yet another table can be maintained to improve the handling of LCTL.

The above mechanism contrasts with the 370-XA mechanism in that there are no intermediate tables in 370-XA interpretive-execution mode. The frequency of references to tables (E) and (F), at a higher cost in machine cycles than for successful references to the intermediate tables (C), increases, but this is counterbalanced by the absence of overhead to maintain the intermediate tables. With a shadow-table approach, the cost in maintenance processing time and table space could have increased remarkably for guests using 31-bit real and/or virtual addresses. The effects would be compounded as more guest applications made more use of multiple address spaces through the use of the dual-address-space (DAS) facility. An additional table structure is not required for the efficient handling of IPTE.

A perspective on the significance of this mechanism (and on shadow tables in the case of VM/370) may be gained by observing that the mechanism is not used for high-performance preferred guests. Since CMS does not use (guest) DAT, neither is the mechanism employed in support of this, the most frequently used on-line environment. On the other hand, for supporting pageable guests that use DAT, the 370-XA mechanism for translating guest virtual addresses not only simplifies and reduces support programming, but is also amenable to implementation using high-performance mechanisms provided in the machine.

- *Multiprocessing*

In significant distinction to previous virtual-machine support, 370-XA interpretive-execution mode provides for full use of both host and guest multiprocessing. The following benefits are realized:

- A high-performance multiprocessing preferred guest can be provided.
- The ability is provided to more thoroughly check out a new or changed multiprocessing operating system when it is executed as a guest system, even using pageable mode.
- Full, effective use is made of multiple host processors. Both overall performance and availability benefit by having such flexibility.

Key capabilities that contribute to the effective use of multiprocessing are the following:

- Prefixing (discussed in the preceding section on storage).
- Interception control of the COMPARE AND SWAP (CS) and COMPARE DOUBLE AND SWAP (CDS) instructions.

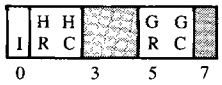

Figure 7 RCP table byte. I = interlock control; HR = saved host reference indicator; HC = saved host change indicator; GR = saved guest reference indicator; GC = saved guest change indicator.

- Interlock control of access to the additional tables used by the key-handling operations (a special handling required in pageable mode only).
- Interlock control of the IPTE instruction.
- Asynchronously settable intervention controls.

The last four items are discussed further in the following sections.

Spin locks

The compare-and-swap instructions are typically used to implement operating-system locks. A lock is used to control serial access to resources. The compare-and-swap instructions resolve any potential race conditions for ownership of the lock and deposit an identifier associated with the requester, task or CPU, on behalf of which the lock is set closed.

One type of lock, called a *spin lock*, poses special problems when used by a guest. By definition, a requester of a closed spin lock loops, waiting for the lock to open. The critical consideration in the original design is that the lock, once closed, is almost never held for a "long" time; i.e., the possible alternatives to spinning would take as much time as the expected remaining hold time. Ordinarily, except for machine checks, the owning CPU is not interruptible until the lock is cleared. However, in a virtual-machine environment, the guest has no control over when one of the real processors is interrupted. Thus, a second processor could be left spinning indefinitely, though this is not likely when ready guests are being regularly redispatched. However, host programs can, if experience in an individual instance indicates excessive time is being lost, gain control to analyze the circumstances by intercepting on the condition code typically set for the closed-lock condition (alternatively, a guest system can be modified to explicitly indicate this condition to the host program).

Key handling

The reference-and-change-preservation (RCP) table, one of the satellite tables designated by the state description, serves a purpose similar to the swap tables provided by VM/370 (see [10]). One byte is provided in the RCP table for each 4K bytes of guest real storage. In each byte there are two change and two reference bits, one each for guest and for host use, and an interlock bit. This is illustrated in Fig. 7. When either the host or the guest modifies the change and reference indicators in the real key associated with an assigned page

frame of host real storage, the old values of these indicators are saved in the RCP byte for the other system, guest or host. For each system, the logically correct status of a page frame is the OR of the current value of the real indicators and RCP byte indicators for that system. The use of RCP information is implicit in the execution of guest key-handling instructions, but must be developed by programmed means in the host. Two of the six key-handling instructions are successfully executed by using only RCP information, instead of causing interception, when a host translation exception prevents access to the real key. This is a significant difference from System/370, where execution of the remaining instructions is also provided, through assists, under this condition. There, execution uses the key in the swap table, which is not provided in the RCP table.

Because two separate locations are referenced (the real key and the corresponding RCP byte) and must be kept consistent, and because separate accesses could be attempted by two different CPUs nearly concurrently in a multiprocessing system, all accesses are normally required to set the interlock control. When execution of a key-handling operation for the guest is attempted, exit from interpretive-execution mode by interception takes place if an interlock is already set. The interlock bit in each byte of the RCP table facilitates two activities.

First, it allows the machine to execute guest key-handling instructions in a guest multiprocessing environment. Natively sequential access is enforced for references to the real storage key, to either ensure consistent updating or (for the RESET REFERENCE BIT [RRB] instruction) to provide a read access followed by a write access without an intervening access, thus ensuring that accurate information is maintained. The existence of the RCP table in interpretive-execution mode introduces the need for additional controls. For example, since a guest SET STORAGE KEY (SSK) instruction must initially obtain the information with which to update the associated RCP byte for the host, it must employ the same interlock on access to the real key as was employed natively by the RRB instruction. The additional RCP interlock bit is used to prevent an intervening access for key information before the RCP byte is updated; it is also used by the INSERT STORAGE KEY (ISK) instruction to ensure use of consistent information. The RCP interlock does not inhibit the channel from concurrently updating the real changes and reference indicators, as must the interlock on the real key.

Second, in a host multiprocessing system, the RCP interlock control allows use of key-handling instructions by the host program on blocks assigned to the guest without suspending execution of that guest on another CPU. Thus the host can "steal" a page from a guest without otherwise disturbing an ongoing execution of that guest. By using the interlock, each system, host and guest, can independently conduct paging activity that affects the same real host page frame. This mechanism is not used in preferred mode since the host makes no attempt to dynamically reassign real host storage assigned to a preferred guest.

The comments in this section apply as well to the extended-key-handling instructions.

INVALIDATE PAGE TABLE ENTRY (IPTE) instruction handling

Besides setting the invalid bit in the designated guest page table entry to one, the IPTE instruction deletes an associated entry in a TLB. Under interpretive execution, however, the TLB may manifest itself in several distinct places. If the guest has been dispatched on several host CPUs, residual guest entries may or may not reside in the TLB of several real CPUs. If guest multiprocessing is being used, a relevant TLB entry may exist in one or more real CPUs for other guest CPUs. In addition, one or more of the guest CPUs may be suspended, awaiting simulation of a function by a host program. The host program may be using the result of translating a guest address, an address possibly affected by the IPTE on another guest CPU.

There are two fundamental problems to be dealt with: a) how the residual entries in TLBs on other CPUs are to be removed, and b) how the interaction with a host program is to be handled. These problems are dealt with differently depending on whether the guest is being treated as a uniprocessor machine or as a multiprocessor machine. Residual TLB entries for a uniprocessor guest are handled as follow:

1. If the real processors are implemented so that guest entries are not retained in the real TLB after leaving interpretive-execution mode, IPTE purges only the TLB of the issuing processor.
2. If the real processors retain residual guest entries after exiting from interpretive-execution mode, then, in addition to purging the TLB of the issuing processor, the real CPU address is stored in the current state description on exit from interpretive-execution mode. When interpretive-execution mode is subsequently entered on any other processor, a mismatch of the stored CPU address and the address of the real CPU causes the TLB to be purged of all guest TLB entries associated with the current state description. As a programming technique, a mismatching value can be deliberately inserted to effectively accomplish a guest PTLB operation, or to induce the equivalent of a guest PTLB instruction when the host has changed guest translation parameters or tables.

The interaction with the host for a uniprocessor guest is relatively simple: since simulation is synchronous for a

uniprocessor guest, residual products of translating guest addresses normally do not exist in the host program domain while that guest is being executed. The host program normally does not need to use the interlock control.

For a multiprocessing guest, two additional capabilities are provided to help handle the additional considerations. Broadcasting can be enabled for guest IPTE instructions, and an interlock control is provided for coordinating the use of translations of guest addresses by host simulation programs on other CPUs; interception is recognized for IPTE when the interlock is set.

Finally, regardless of whether or not multiprocessor guests in addition to uniprocessor guests are being supported, a host IPTE purges not only the designated host entry but also all existing guest entries which have a dependency on the host entry being removed (normally accomplished by deleting all guest entries from the real TLB). Whether or not additional interlocks within the host control program are needed depends on the particular host control program; in general, the usual considerations apply.

Intervention requests

Interruption requests arising from conditions external to the guest program are kept pending in the form of bits in the state description. They are used to signal the availability of guest I/O-interruption information, which may have been received from the host channel on another host CPU, to signal an external interruption condition, and to signal an operator request to STOP the guest CPU. Collectively, they are called the *intervention controls*. In contrast with most other controls, these bits may be set asynchronously by one CPU while interpretive execution is in progress on another CPU using the associated state description, with the assurance that the new setting will be observed on behalf of the running guest. As a consequence, host-to-host CPU signaling to obtain the attention of the appropriate real CPU that is handling the affected guest is not usually needed. This avoids disrupting the ongoing execution of a guest when the guest is disabled for the corresponding interruption. It also avoids using interception to monitor guest events for changes to the enabled state.

The intervention controls are necessarily examined under two conditions: a) when a guest operation is executed that enables the guest for the corresponding interruption, or b) periodically, at least for request types for which the guest is enabled. Since the time elapsing between periodic inspections is model dependent, responsiveness to the condition that a request for an interruption has been made pending differs from native execution. However, since this period is chosen with reasonable timing variations in I/O activities for a model taken into consideration, any discrepancy with native execution should have no significant effect.

Conclusions

The 370-XA interpretive-execution architecture makes available for use by guest systems most of the facilities of both the System/370 and the 370-XA architectures. Machine resources are used relatively efficiently, migration of applications to the new architecture and use of the new mode of the machine is facilitated, performance is generally improved, complexity is reduced in both the machine and the supporting control program, less time and fewer resources are needed in making new machine functions also available in a virtual machine, and performance disparities among a variety of guests are diminished. Engineers have greater latitude in adapting the characteristics of different machine designs to meet the architecture. Where more than one control program is expected to act as a host, an additional reduction is realized in what might otherwise be essentially redundant support programming. Thus, efficiencies are achieved at the same time a more generally usable capability is provided. These characteristics are expected to encourage an expansion in the use of virtual-machine capabilities, including in particular the further development of applications as virtual machines (i.e., subsystems) intended to run only in the guest-host environment.

Acknowledgments

The author participated in the design of the preferred-machine assist, but the other System/370 assists, the VM/370 control program, and the measures of performance, all of which are summarized in the first part of the paper, are the work of others (see the references). Work on 370-XA was initiated and completed under the direction of A. Padegs. He provided crucial guidance, insights, and encouragement over several years. P. H. Tallman was instrumental in the development of VMA originally, and subsequently made significant technical contributions to the 370-XA interpretive-execution architecture, in addition to providing counsel, as did T. O. Curlee, throughout the development process. R. M. Smith contributed invaluable technical advice on several occasions. It is fitting and a privilege to recognize the indebtedness to these colleagues, as well as many not named here, for their valuable contributions.

References and note

1. D. Rosen, "The Works," *Think* **21,** 1, 32–36 (1982).
2. *IBM Virtual Machine Facility/370: Remote Spooling Communications Subsystems (RSCS) User's Guide*, Order No. GC20-1816, available through IBM branch offices.
3. R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM J. Res. Develop.* **25,** 5, 483–490 (1981).
4. *IBM Syst. J.* **18,** 1 (1979). This issue is devoted to articles concerning Virtual Machine Facility/370. Extensive additional bibliographic information is included.
5. *IBM System/370 Principles of Operation*, Order No. GA22-7000, available through IBM branch offices.
6. A. Padegs, "System/360 and Beyond," *IBM J. Res. Develop.* **25,** 5, 377–390 (1981).

7. A. Padegs, "System/370 Extended Architecture: Design Considerations," *IBM J. Res. Develop.* **27**, 3, 198–205 (1983).
8. R. L. Cormier, R. J. Dugan, and R. R. Guyette, "System/370 Extended Architecture: The Channel Subsystem," *IBM J. Res. Develop.* **27**, 3, 206–218 (1983).
9. *IBM 370-XA Principles of Operation*, Order No. GA22-7085, available through IBM branch offices.
10. *IBM Virtual Machine Facility/370 Introduction*, Order No. GC20-1800, available through IBM branch offices.
11. *IBM Virtual-Machine Assist and Shadow-Table-Bypass Assist*, Order No. GA22-7074, available through IBM branch offices.
12. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide, Volume 1, Appendix A*, Order No. LY20-0892, available through IBM branch offices.
13. R. A. MacKinnon, "The Changing Virtual Machine Environment: Interfaces to Real Hardware, Virtual Hardware, and Other Virtual Machines," *IBM Syst. J.* **18**, 1, 18–46 (1979).
14. *IBM Airline Control Program/Transaction Processing Facility ACP/TPF, Hypervisor Program Reference*, Order No. GH20-2311, available through IBM branch offices.

Received July 30, 1982; revised June 10, 1983

Peter H. Gum *IBM Information Systems and Technology Group, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. Gum is a senior programmer in the Central Systems Architecture Department at Poughkeepsie. He joined IBM in 1964 in Poughkeepsie as a system programmer working on the operating system for the IBM System/360, and subsequently participated in the design of several versions of the control program. In 1973 he joined Central Systems Architecture, where he participated in the design of extensions to the architecture of the IBM System/370. He received a B.A. from Oberlin College, Oberlin, Ohio, in 1958 and an M.A. from the American University, Washington, D.C., in 1962, both in mathematics. He received an IBM Outstanding Contribution Award for his work on system control programs and an IBM Division Award for his work on the architecture of System/370. Mr. Gum is a member of the Association for Computing Machinery.