Reduction of 2-0-Translation Table Maintenance Overhead in a Virtual Machine System

HIDENORI UMENO*, TAKASHIGE KUBO* and SHIGEO TAKASAKI*

Methods are presented to reduce CPU overhead when an Operating System (OS) with multiple virtual storage is running under a Virtual Machine System (VMS) with 2-0-translation tables, which are also called shadow tables and translate level 2 addresses (i.e. OS's virtual addresses) to level 0 addresses (i.e. system's real addresses).

The conventional multiple 2-0-translation table method has the advantage that 2-0-translation tables can be used corresponding to each OS's virtual storage, and it reduces VMS's CPU overhead due to an OS switching its virtual storage.

However, it has some disadvantages. That is, (1) an OS in a VM may issue a privileged instruction to invalidate pages during its paging process. In order to simulate it, the associated 2-0-translation table entries have to also be invalidated, which means to cancel the address mapping from level 2 page addresses to level 0 page addresses. This invalidation overhead will increase in proportion to the number of 2-0-translation tables. (2) Moreover, the switching of multiple 2-0-translation tables is a new additional CPU overhead.

In order to solve the problem, an assist is invented to selectively invalidate the 2-0-translation table under the condition that a page address to be invalidated is given in a privileged instruction used for the OS's page invalidation process. Moreover, another assist is developed to realize fast 2-0-translation table switching corresponding to switching OS's virtual storage. These two assists are named, generically, Multi-Shadow Assist (MSA), which is applicable to any Virtual Machine (VM) irrespective of its memory attributes.

Acceleration ratios of MSA to associated software processes are estimated at 3.4 ~ 24.4. Multi-Shadow Assist is experimentally implemented in Hitachi's VMS using conventional hardware architecture, and performance measurements are obtained on some benchmark jobs for this report.

Performance data which confirm the effectiveness of MSA are presented. Multi-Shadow Assist reduces almost all the 2-0-translation table maintenance overhead, with a 10% reduction in mean instruction execution time, and a 40% ~ 60% reduction in total CPU overhead.

Moreover, a method to prevent the double invalidation of the 2-0-translation tables is proposed. It is effective for the temporary increase of the number of the OS's active virtual storage.

This performance improvement efficiently supports the OS with multiple virtual storage running under the VMS using conventional hardware architecture.

1. Introduction

A Virtual Machine has a functionally similar architecture to a real host computer on which a statistically dominant subset of virtual processor instructions executes directly[1),2)]. A Virtual Machine System (VMS) consists of more than one Virtual Machine (VM) which can run concurrently. A Virtual Machine Monitor (VMM) is a control program of a VMS.

Each user of a VMS can select a different Operating System (OS) because different OSs can run concurrently in different VMs. Virtual Machines are usually used for system development and testing, and to obtain the services of more than one OS from a single real host computer.

The practical use of a VMS is dependent on its CPU overhead and so, many attempts have been made to reduce this overhead[3)-6)]. Representative examples of these are Virtual Machine Assist (VM-Assist)[3)], Virtual Equal Shadow method (VES)[5)], and Multiple 2-0-Translation Table method[4)].

A 2-0-translation table is an address translation table which translates level 2 addresses (i.e. OS's virtual addresses) to level 0 addresses (i.e. system's real addresses). A 2-0-translation table is also called a shadow table.

The VM-Assist is a microcode implementation designed to enhance the execution of privileged instructions and supervisor calls associated with VMs. In most cases the effects of VM-Assist are significant, and it reduces the VMM's CPU overhead by about 80% for some benchmark jobs[3)]. The VES and Multiple 2-0-Translation Table support have reduced the VMS's CPU overhead caused by paging/swapping processes in the multiple virtual storage of an OS under a VMS.

In spite of these efforts, the VMS's CPU overhead

*Systems Development Laboratory, Hitachi Ltd.

for an OS with multiple virtual storage is not low enough yet, when using conventional hardware architecture. The reasons are as follows.

(1) The VES can be applied only to one VM (i.e. V=RVM: explained later). Namely, the performance can be enhanced only for V=RVM[5].

(2) The Multiple 2-0-Translation Table method can avoid the full invalidation of a 2-0-translation table by switching the 2-0-translation table corresponding to the OS switching its virtual storage. This considerably reduces the VMS's CPU overhead.

However, it has some disadvantages. That is, an OS in a VM may issue a privileged instruction to invalidate its own pages during its paging process. In order to simulate it, the associated 2-0-translation table entries have to be also invalidated, that is, the address mapping defined by the entries from level 2 page addresses to level 0 page addresses is to be invalidated. The invalidation overhead will increase in proportion to the number of 2-0-translation tables.

Moreover, the switching of the 2-0-translation table is a new additional CPU overhead. This new additional CPU overhead may cancel the original effects of this method.

As another reduction method, Fast I/O Simulation for V=Resident VM is presented[6]. It has considerably reduced VMS's I/O simulation overhead. However, it does not address the reduction of the 2-0-translation table maintenance overhead for an OS with multiple virtual storage under a VMS.

Several reduction methods are proposed here for the VMS's CPU overhead for an OS with multiple virtual storage under a VMS using conventional hardware architecture. They are as follows.

(1) Multi-Shadow Assist (MSA), which implements selective invalidation and fast switching of the 2-0-translation tables. This solves the above-mentioned disadvantages of the multiple 2-0-translation table method.

(2) Prevention of 2-0-translation table double invalidation.

This paper consists of the following sections.

Section 2: The concept of a Virtual Machine System is explained.

Section 3: The conventional reduction methods for VMS's CPU overhead are described, the multiple 2-0-translation table method, invalidation and validation are explained in detail.

Section 4: The new reduction methods for 2-0-translation table maintenance are described.

Section 5: Performance data will be presented and it will be shown that VMS's performance is sufficiently improved to make practical the use of any VMs with multiple 2-0-translation tables.

Section 6: Conclusions.

2. Virtual Machine System (VMS)

A Virtual Machine System gives multiple users execution environments that have an architecture similar to a real host computer. The conventional hardware architecture of a real host computer, as discussed here, has the following characteristics[7].

(1) Central Processing Unit (CPU)

(a) Two processor states, called a supervisor state and a problem state.

(b) Privileged instructions, which can be executed only in the supervisor state.

(c) A dynamic address translation (DAT) feature, which translates a virtual memory address into a real memory address. The DAT feature uses address translation tables called Segment Tables/Page Tables (STs/PTs). The DAT feature can perform only one level address translation, namely it can not do two level address translation.

(d) Translation Look-aside Buffer (TLB), which contains pairs of virtual page addresses and real page addresses. TLB is used for fast address translation by the DAT feature.

(e) Buffer Memory, which contains copies of the contents of the real memory. This is used for fast access to data.

(f) A prefix area, which is in real memory and contains control information for various hardware interruptions.

(2) Virtual Machine

Virtual machines also have the two processor states, complete instruction sets, storage, and prefix areas described above for CPUs. The mechanisms used by VMM to implement a VM fall into three categories: processor simulation, memory simulation and I/O simulation.

Processor simulation is accomplished by running VM programs on a real processor. Most of the nonprivileged or privileged state instructions are executed directly or are emulated by the processor, respectively. Some sensitive instructions are trapped and simulated by VMM.

Memory simulation is performed by providing a virtual memory to each VM. This virtual memory appears to be a dedicated real memory complete with address translation hardware. An OS in a VM manages this memory as if it were real. Therefore, when an OS in a VM is a virtual storage OS (VOS), a 3-level memory hierarchy is constructed, where:

level 0 memory: real memory of a real processor;

level 1 memory: memory of a VM, (An OS in a VM regards this as real memory);

level 2 memory: virtual storage created by VOS in a VM.

Simulation of the dynamic address translation feature is performed by the VMM using real hardware and special address translation tables (called 2-0-translation tables, or shadow tables) that map level 2 memory

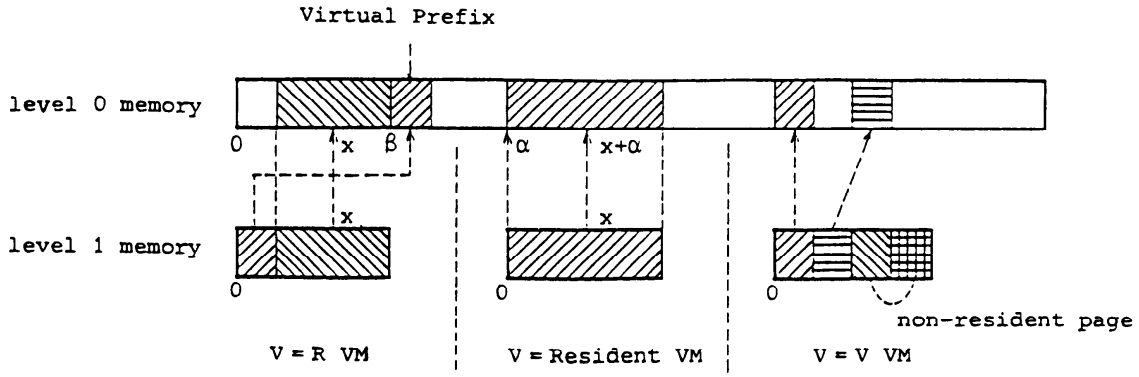


Fig. 1 VM's memory attributes.

addresses into level 0 memory addresses.

According to their memory attributes three kinds of VMs are defined as illustrated in Fig. 1.

V=R VM: The level 1 memory is resident in the real memory, and level 1 memory address=level 0 memory address except in the virtual prefix area.

V=Resident VM: The level 1 memory is resident in the real memory, and level 0 memory address=level 1 memory address+α, where, α ($\neq 0$) is a fixed page number given to each V=Resident VM at the VMS's system generation.

V=V VM: The level 1 memory is virtual memory to VMM, it is pageable, and is not always resident.

Address translation tables are defined as follows.

1-0-Translation Tables: Address translation tables which translate level 1 memory address to level 0 memory address. These are also called 1-0-Translation Segment Tables and Page Tables.

2-1-Translation Tables: Address translation tables which translate a level 2 memory address to a level 1 memory address. These are also called 2-1-Translation STs and PTs. 2-1-Translation Tables are created and managed by an OS in a VM, therefore, they may also be called the OS's translation tables.

2-0-Translation Tables: Address translation tables which translate level 2 memory addresses to level 0 memory addresses. These are also called 2-0-Translation STs and PTs. 2-0-Translation Tables are created and managed by VMM and VM-Assist, and used by the DAT feature when an OS is running in a level 2 memory. Their entries are composed from 2-1-Translation Table entries and 1-0-Translation Table entries. 2-0-Translation Tables are also called Shadow Tables.

3. Conventional Reduction Methods for CPU Overhead of VMS with 2-0-Translation Tables

3.1 Conventional Reduction Methods

The representative methods to reduce VMS's CPU overhead are listed below.

(1) Virtual Machine Assist (VM-Assist): This is a microcode assist for the frequently used privileged instruction simulation. VM-Assist's effects are significant as described in Sec. 1. It also assists creation and validation of 2-0-translation page table entries.

(2) Virtual Equal Shadow (VES): For the V=R VM, 2-0-translation tables can be eliminated as illustrated in Fig. 2. The Operating System's translation tables (i.e. 2-1-translation tables) are also used as the 2-0-translation

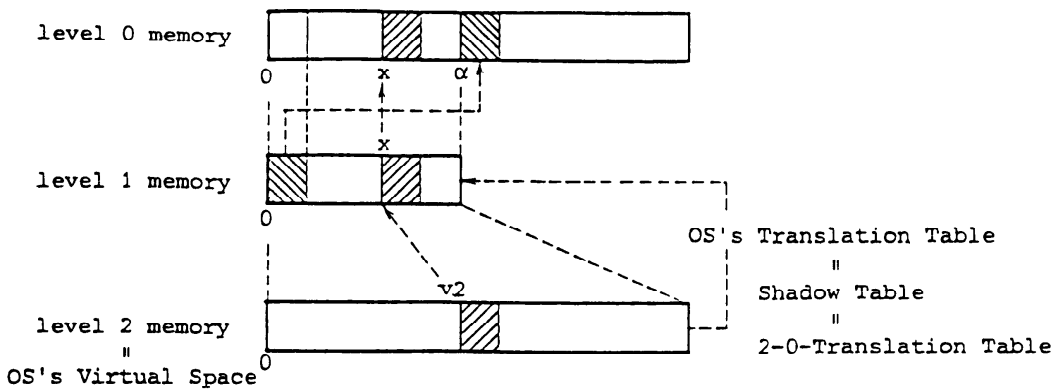


Fig. 2 Virtual Equal Shadow for V=R VM.

tables as is, and this significantly reduces 2-0-translation table maintenance overhead[5].

(3) Multiple 2-0-Translation Table method: An OS with multiple virtual storage in a VM can be given multiple 2-0-translation tables corresponding to each the virtual storage of an OS. This is called a multiple 2-0-translation table method. In a single 2-0-translation table method, there is only one 2-0-translation table in a VM.

The advantages of the multiple 2-0-translation table method are explained as follows. See Table 2, (#3). (Table 2 is explained in the later section).

An OS in a VM issues a privileged instruction to switch its virtual storage. In a single 2-0-translation table method, all the table entries have to be invalidated (this is called a full invalidation) to simulate it, because the address mapping from the OS's virtual page addresses (i.e. level 2 page addresses) to the OS's real page addresses (i.e. level 1 addresses) is entirely changed.

The invalidation of an address translation table entry means that the address mapping defined by that entry is canceled. The full invalidation of an address translation table means that all the table entries are entirely canceled.

In the multiple 2-0-translation table method, each OS's virtual storage is given its own 2-0-translation table. Therefore, it is enough only to switch the 2-0-translation table corresponding to the OS switching its virtual storage. Thus, the full invalidation of a 2-0-translation table can be avoided. This considerably reduces the invalidation overhead of the 2-0-translation table. Moreover, the ratio of the number of the references not in TLB to that of the total references (i.e. Not is TLB Ratio: NITLBR) is reduced by the multiple 2-0-translation table method, because it can avoid the full TLB purge due to the OS switching its virtual storage.

A full 2-0-translation table invalidation causes TLB purge of all its entries and brings an NITLBR increase. This exerts a bad influence upon the MIET (i.e. Mean Instruction Execution Time) of the real host computer. Therefore, a full invalidation has to be avoided and a selective invalidation has to be adopted as often as possible.

Table 1 shows which conventional reduction methods are effective with what kinds of OS in a VM. There are three kinds of OS as explained here.

Table 1 Effectiveness of conventional reduction methods.

Reduction Methods \ OS	Real Storage	with single virtual storage	with multiple virtual storage	Remarks
VM-Assist	○	○	○	○: small/ medium ◎: large ×: none
Virtual Equal Shadow	×	○	◎	
multiple 2-0-Trans. Table	×	×	◎	

Real Storage OS: An OS which does not have virtual storage.

An OS with a single virtual storage: An OS which has only one virtual storage.

An OS with multiple virtual storage: An OS which has multiple virtual storage.

This table shows that the Virtual Equal Shadow method and the Multiple 2-0-Translation table method are more effective with an OS with multiple virtual storage than with any other OS. The reason is that 2-0-translation table maintenance overhead is dominant in the total CPU overhead of an OS with multiple virtual storage under a VMS, and the two methods are effective especially for 2-0-translation table maintenance overhead reduction. On the other hand, the 2-0-translation table maintenance overhead is small in the other OSs.

For a drastic reduction of the overhead it is desirable to eliminate 2-0-translation tables. However, the elimination is applicable only for the V=R VM (i.e. VES method described above) under the hardware architecture described in Sec. 2. On the other hand, the multiple 2-0-translation table method is effective with any type of VM memory attributes.

This paper assumes that the above-mentioned conventional reduction methods have already been taken.

Table 2 shows 2-0-translation table maintenance overhead and OS events which cause that CPU overhead. It also shows 2-0-translation table maintenance overhead ratios per one time in (#4). Here, the 2-0-translation table maintenance overhead is explained for further discussion with Table 2.

3.2 2-0-Translation Table Maintenance Overhead

2-0-Translation table maintenance consists of the following processes.

(1) Acquisition, release and initialization of the 2-0-translation table area.

(2) 2-0-Translation page table invalidation.

(3) 2-0-Translation page table entry validation and creation.

(4) 2-0-Translation table switching.

In these processes, process (1) is executed only once at the VM's log on/off time. Therefore, its overhead may be ignored. However, processes (2), (3) and (4) are processed while the OS is running under a VMS, and they may cause a relatively high CPU overhead.

Process (2) is explained as follows.

When an OS's virtual pages are invalidated at page release, page out, segment release, swap out and so forth (See Table 2, #1, #2), the OS's page translation table entries for the translation from level 2 memory to level 1 memory are partly invalidated. Therefore, the associated or entire entries of the 2-0-translation tables must be invalidated. This is called a selective invalidation or a full invalidation, respectively.

This overhead increases in proportion to the number of invalidated entries or 2-0-translation tables. An OS's

Table 2 2-0-Translation table maintenance overhead vs relative OS events.

#	OS events \ 2-0-trans. tab. m. o.	selective invalidation	full invalidation	validation & creation	2-0-trans. tab. switch	double invalidation	NITLBR increase	priv. inst.
1	page invalidatn page out	○ (S, M)	○ (S, M)	○ (S, M)	—	— (S) ○ (M)	○ (S, M)	PPTLB*1
2	segment release swap out	—	○ (S, M)	○ (S, M)	—	— (S) ○ (M)	○ (S, M)	PTLB*2
3	virtual storage switch	—	○ (S) — (M)	○ (S) — (M)	— (S) ○ (M)	—	○ (S) — (M)	LCTL (CR1)*3
4	overhead ratios per one time	0.8 (S, M) (one entry)	90 (S) 90·n (M)	1 (S, M) (one entry)	— (S) (1+0.06·n (M)	— (S) 90·m (M)	high (S) low (M)	—

remarks

○: cause this overhead
—: does not cause this overhead
S: Single 2-0-trans. table method
M: Multiple 2-0-trans. table method
n: the number of 2-0-translation tables
m: the number of unused 2-0-translation tables
*1 Partial Purge TLB
*2 Purge TLB
*3 Load Control (CR1)

segment release, swapping out and log-off (Table 2, #2) do not take place so often. Therefore, that event will be ignored, hereafter.

Process (3) is explained as follows.

When a 2-0-translation page translation exception happens during the OS's running in a VM, the 2-0-translation page table entry is composed from the contents of the OS translation table (i.e. 2-1-translation table) entry and those of the 1-0-translation table entry. Conventionally, this process (3) is achieved by one of the VM-Assist's features. This overhead increases in proportion to the number of referred virtual pages.

Process (4) is explained as follows.

When an OS with multiple virtual storage running under a VM switches its virtual storage, the corresponding 2-0-translation table is searched and is also switched in the multiple 2-0-translation table method. On the other hand, in a single 2-0-translation table method, all the 2-0-translation table entries are entirely invalidated, as described in the preceding section. (See Table 2, #3). This switching overhead increases in proportion to the number of 2-0-translation tables.

Table 2 also shows the disadvantages of the multiple 2-0-translation table method. They are described here as follows. See Table 2 (#4).

3.3 Disadvantages of the Multiple 2-0-Translation Table Method

3.3.1 Full Invalidation of the 2-0-Translation Tables

An OS invalidates the page table entry and clears the TLB of the associated entries during the paging process (See Table 2, #1). For this purpose, an OS usually uses a privileged instruction, which may have one of the following functional specifications.

(1) Specification of virtual page address to be invalidated.
(2) Specification of real page address to be invalidated.

An OS in a VM does not recognize any 2-0-translation table, but only recognizes its own translation table (i.e. 2-1-translation table). A privileged instruction with virtual page specification (1) specifies a level 2 page address in a VM environment. Therefore, the associated 2-0-translation page table entry is easily found and invalidated. Moreover, the associated TLB entries can be invalidated easily. That is, a selective invalidation is carried out. In this case, the simulation steps are relatively few, and the influence upon Mean Instruction Execution Time (MIET) after finishing the privileged instruction simulation with virtual specification (1) is relatively small because the selective TLB purge does not cause a "Not in TLB Ratio" increase.

In a VM environment privileged instruction with real page specification (2) specifies a level 1 page address which is translated to a level 0 page address easily. However, because an OS does not specify any 2-0-translation table entries, it is difficult for the VMM to find the associated 2-0-translation page table entry. That is, all 2-0-translation page table entries must be looked up in order to determine whether they will contain the specified level 0 page address. Therefore, it will take many very complex execution steps to make the precise simulation for the privileged instruction with the real specification (2).

In order to avoid this complexity, all 2-0-translation table entries and all TLB entries may be invalidated for the privileged instruction with the real specification (2). That is, a full in-

validation method is a very simple simulation method, however, its execution steps are in proportion to the number of 2-0-translation tables, (See Table 2, #4), and it has a bad effect upon MIET after the privileged instruction simulation with real specification (2), because of the full TLB purge.

Here, the specifications (1) and (2) are examined quantitatively. The examination supposes that a full invalidation method is taken conventionally for the real specification (2). Table 3 shows the ratio of the number of invalidated 2-0-translation table entries, the ratio of the simulation execution steps and the ratio of the number of invalidated TLB entries. This table reveals that every item of the real specification (2) is from tens of times to thousands of times greater than that of the virtual specification (1). Therefore, the frequent use of a privileged instruction with real specification (2) will bring high VMS CPU overhead.

Accordingly, under a VMS a privileged instruction with virtual specification (1) is more desirable than a privileged instruction with real specification (2).

Conversely, under a real computer system the latter is more desirable than the former, because the TLB entry purge for the latter can be more selective than that for the former, because the same virtual page address of different virtual storage may be mapped to different real page addresses. For example, virtual page address V of virtual storage S1, S2 is mapped to R1, R2, respectively. That is,

(a) V in S1→R1,

(b) V in S2→R2.

The former invalidates both (a) and (b), however, the latter invalidates only either (a) or (b). For this reason, the real specification (2) is used in most of the native OSs. This is also true in the VMS; however, in the simulation of VMS, it is difficult for VMM to find the associated 2-0-translation table entry in the real specification (2) as described above. Therefore, any selective invalidation is difficult to execute, and conventionally a full invalidation is carried out for the real specification (2).

As a privileged instruction with real specification (2), the Partial Purse TLB (PPTLB) instruction exists. This instruction specifies a page table origin and a page index and invalidates the page table entry which contains a real page address (R) to be invalidated, and clears the TLB of the associated entries which contain the real page address (R). Conventionally, in the VMS the full invalidation method has been taken for this PPTLB simulation. The reduction method for it will be described in Sec. 4.

3.3.2 OS's Virtual Storage Switch under VMS

An OS issues a privileged instruction (Load Control (CR1): This loads a segment table origin into Control Register (CR1)) to switch virtual storage. See Table 2, (#3). The Load Control (CR1) simulation consists of two processes. One is to load a new 2-1-translation segment table origin into the virtual CR1. The other is to switch 2-0-translation tables corresponding to the new virtual CR1.

Moreover, if the associated 2-0-translation table can not be found, and additional 2-0-translation tables can not be created because of memory limitations, a 2-0-translation table for a different virtual storage will be stolen. This stolen table is thoroughly invalidated before reuse. This process brings high CPU overhead, as mentioned in the preceding section.

Therefore, the memory limitations must be extended so that additional 2-0-translation tables can be made when necessary. Moreover, because the frequency of an OS's Load Control (CR1) issuance is very high, the 2-0-translation table search process must be fast. The methods will be described in Sec. 4.

3.3.3 2-0-Translation Table Double Invalidation

When the number of active virtual storage of an OS increases from N_1 to N_2 ($N_1 < N_2$), the number of corresponding 2-0-translation tables is easily increased from N_1 to N_2. The reason is that new 2-0-translation tables may be created as far as the system allows, if the corresponding 2-0-translation table can not be found.

After that, when the number of active virtual storage of an OS decreases to N_1, the Purge TLB instruction is issued to clear the TLB of all its entries. However, it is difficult to decrease the number of 2-0-translation tables to N_1, because it is difficult for VMM to find the corresponding 2-0-translation tables to be inactivated. An OS does not inform the VMM of its invalidated virtual storage identifications. For this reason, VMM invalidates, overall, N_2 2-0-translation tables for the

Table 3 Virtual page specification (1) and real page specification (2).

#	Item \ Specification	(1)	(2)	Remarks
1	Number Ratio of Invalidated 2-0-Trans. Table Entries	1	4096	N: number of 2-0-Trans. Tables Segement Size: 64KB Page Size : 4KB Virtural Space Size : 16MB (1): selective invalidation (2): full invalidation
2	Ratio of Simulation Execution Steps	1	$7*N+4$ when $N \leqq 10$	
3	Number Ratio of Invalidated TLB Entries	Max N	All TLB Entries	

Purge TLB simulation. However, it is better that they remain as 2-0-translation tables for subsequent reuse rather than that they be released.

If the OS continues to run within N_1 active virtual storage after this, N_2-N_1 2-0-translation tables which are already invalidated will not be used at all. Under this condition, if the OS issues the Purge TLB instruction, entirely unused N_2-N_1 2-0-translation tables will also be invalidated again. This means double invalidation of N_2-N_1 2-0-translation tables. The execution steps will increase in proportion to N_2-N_1 as shown in Table 2, (#4).

Therefore, the temporal increase of the number of 2-0-translation tables will bring high CPU overhead even though the frequency may be low. The prevention method for this event will be described in Sec. 4.

4. New Reduction Methods for 2-0-Translation Table Maintenance Overhead

The description in Sec. 3 makes it clear that the following improvements are needed in order to make the best use of the advantages of the multiple 2-0-translation table method and to decrease its disadvantages.

(1) Selective 2-0-translation table invalidation
(2) Fast 2-0-translation table switching
(3) Prevention of 2-0-translation table double invalidation

These improvement methods are described below.

4.1 Selective 2-0-Translation Table Invalidation

Suppose that an OS uses the PPTLB instruction described in Sec. 3.3.1 for the paging process. The following handshaking is introduced in order to change the PPTLB simulation from a full invalidation to a selective invalidation. Namely, "an OS in a VM has to give a virtual page address to be invalidated in the use of a privileged instruction for its page invalidation process."

The PPTLB instruction specifies page index as described in Sec. 3.3.1, however, it does not specify segment index. That is, it does not specify a virtual page address to be invalidated. Therefore, it is required that a correct segment index should be specified in its operand at this handshaking. An OS should recognize a virtual page to be invalidated. Therefore, the above-mentioned handshaking is a reasonable requirement for an OS.

Under this handshaking, the level 2 page address (V2) to be invalidated is described. The V2 consists of correct segment index (SI) and page index (PI). Besides, the level 1 page address (V1) is also obtained from the specified 2-1-translation page table entry. In order to decrease the number of invalidated entries as much as possible, the following simulation method is taken. See Fig. 3.

(1) The level 1 page address (V1) is translated to a level 0 page address (V0) by looking up a 1-0-translation table.
(2) The 2-0-translation page table entry associated with the level 2 page address (V2) is obtained. It is determined from the 2-0-translation segment table origin (STO1), segment index (SI) of (V2), associated 2-0-translation page table and page index (PI) of (V2).
(3) It is determined whether or not the 2-0-translation page table entry contains the level 0 page address (V0).
(4) When it contains the level 0 page address (V0), the 2-0-translation page tabel entry is invalidated. Moreover, the TLB is also cleared of the associated entries which contain the address translation pair (V2, V0). This is carried out by issuing a PPTLB instruction for the associated 2-0-translation page table origin and level 2 virtual page (V2).
(4′) If it does not contain (V0), no entries of 2-0-translation page tables or TLB are invalidated at all.
(5) The above-mentioned (2), (3), (4) and (4′) are performed for all 2-0-translation tables, because common virtual pages, which correspond to the same real pages, may exist.

The maximum number of 2-0-translation page table entries invalidated is equal to the number of 2-0-translation tables. This may happen for common pages of each virtual storage. This maximum number is 1/4096 as compared with a full invalidation method (as shown in Table 3). Moreover, the number of the invalidated TLB entries becomes very small because of the coincidence condition (4) described above.

An assist is developed, which is a microcode implementation to emulate the above-mentioned PPTLB simulation of selective invalidation. This is called a PPTLB-Assist. In it, step (4), described above, is achieved by TLB entry purge micro-instructions.

4.2 Fast 2-0-Transalation Table Switching

A microcode assist which assists the privileged instruction (Load Control (CR1)) simulation is developed. This microcode assist searches for an associated 2-0-translation table, and if the table is found, loads the 2-0-translation segment table origin into a real control reg. 1. If not found, an interruption will take place and control is returned to VMM, because of the low frequency. This assist is called a Load Control (CR1)-Assist.

The above-mentioned PPTLB-Assist and Load Control (CR1)-Assist are named generically Multi-Shadow Assist (MSA). It is applicable to any kind of VM memory attributes (i.e. V=R, V=Resident, V=V, See Fig. 1) and its effects do not depend upon them.

4.3 Prevention of 2-0-Translation Table Double Invalidation

In order to prevent double invalidation of the 2-0-translation tables, it is most desirable that VMM manages whether a valid 2-0-translation table entry is made after a full 2-0-translation table invalidation. However, a valid 2-0-translation table entry is made with the conventional microcode assist VM-Assist.

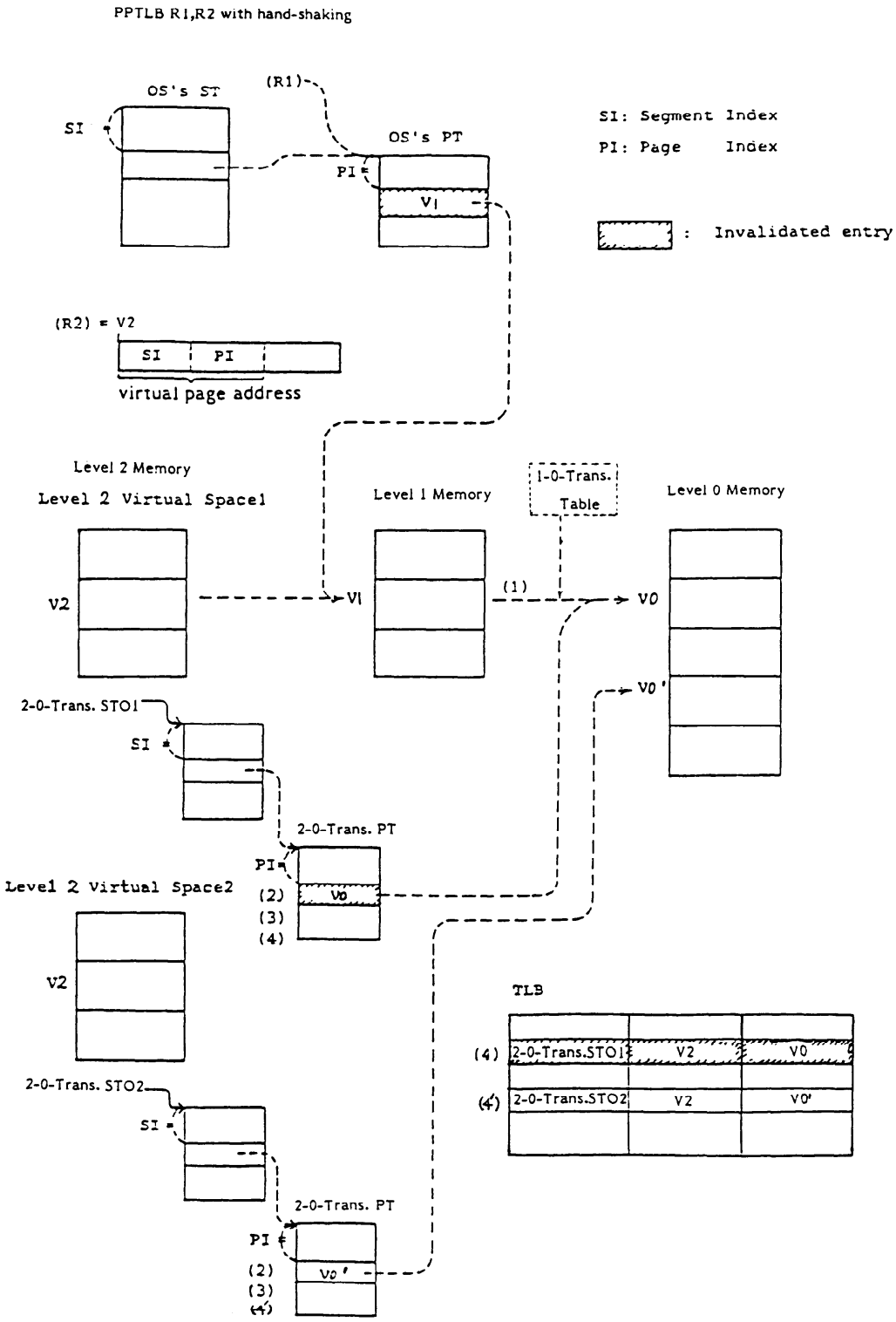


Fig. 3 PPTLB simulation of selective invalidation.

Therefore, it is difficult for VMM alone to manage that information.

Thereupon, the following method is proposed.

Before a 2-0-translation table may be used, it should be selected first by a Load Control (CR1) simulation which switches an OS's virtual storage. When a 2-0-translation table is selected through this simulation, a selection flag (S), which is provided to each 2-0-translation table, is set "ON".

Only when its selection flag is "ON", is there a possibility that the 2-0-translation table may be used and its valid entry may exist. Therefore, it is only needed for the Purge TLB simulation to purge the 2-0-translation tables with the selection flags "ON". This will prevent the double invalidation of the 2-0-translation tables. Naturally, the Load Control (CR1)-Assist should support this selection flag.

The performance improvement methods described in Sec. 4 have been experimentally implemented in Hitachi's Virtual Machine System. The measurement of their effects will be described in Sec. 5.

5. Performance Improvement Effects

5.1 Acceleration Ratios of Multi-Shadow Assist

The ratios of software simulation time to MSA's execution time are called the acceleration ratios of MSA. Their graphs are drawn in Fig. 4. It reveals that the acceleration ratio of PPTLB-Assist is about 3.4 when the number of 2-0-translation tables (N) equals 6, while that of Load Control (CR1)-Assist is about 24.4 when $N=3$.

As for PPTLB-simulation, conventionally a full invalidation method was adopted. The selective invalidation method described in Sec. 4 decreases the software simulation time for the full invalidation method. PPTLB-Assist is a microcode assist for the selective invalidation method. Therefore, its acceleration ratio is to the software selective invalidation simulation time.

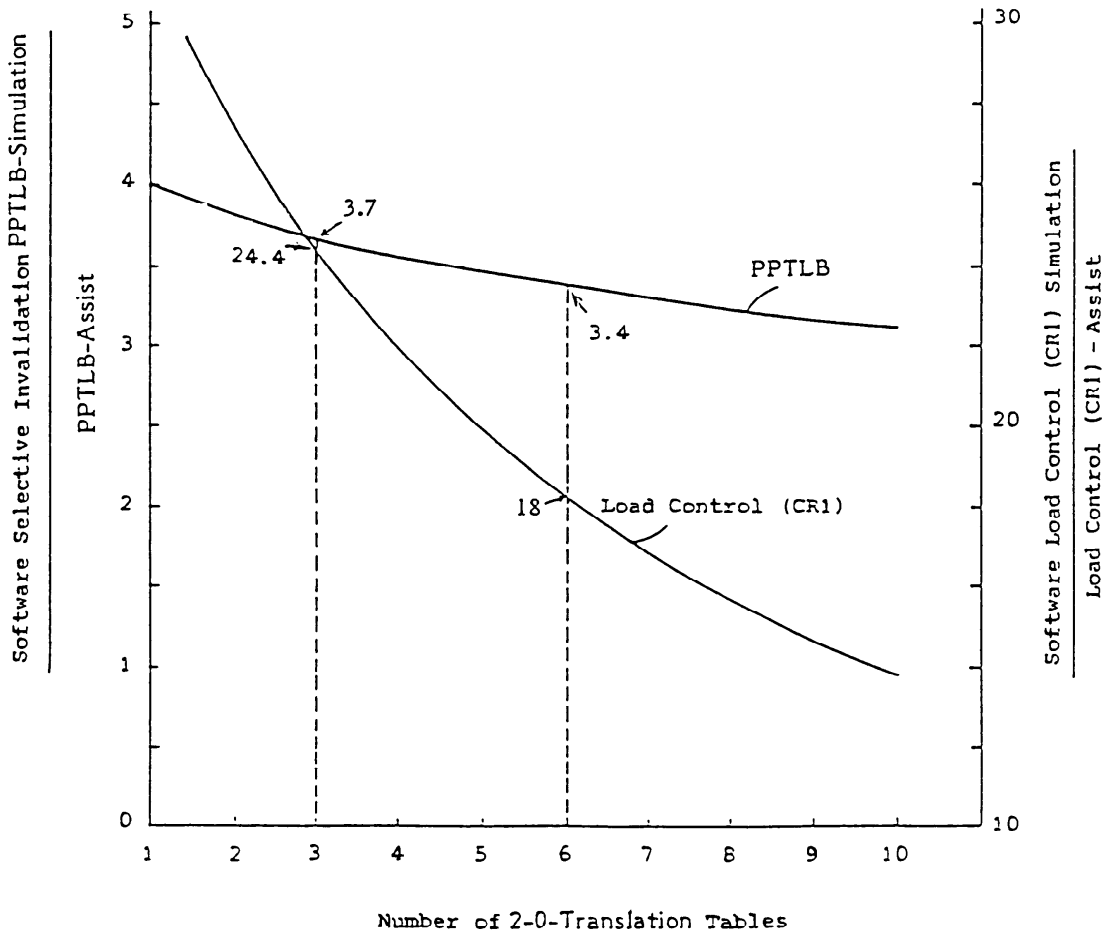


Fig. 4 MSA's Acceleration Ratios.

Table 4 Benchmark jobs.

Job No.	Measurement Conditions	Job Name	CPU Utilization	SIO (F)/ Mega-steps	Remarks
1	OS with multiple virtual Storage under V=Resident 8MB Single VM	*1DMBS Start	2.7%	86.5	*1: Data Base Manage. System *2: Batch Message Processing Program *3: 5-Multi. Jobs
2		*2BMPP×5*3	6.1	42.7	
3		DMBS Start ~BMPP×5	4.6	54.5	

Though the graph is not drawn, PPTLB-Assist is about 140 times faster than the software simulation time of the full invalidation method when $N=6$.

Thus, the software selective invalidation method reduces the full invalidation PPTLB-simulation time. Moreover, PPTLB-Assist reduces the software selective invalidation PPTLB-simulation time. These data are obtained from the experimental (micro-)program coding.

5.2 Performance Evaluation Tools

The following tools are used.

(1) Software monitor: This is built in the VMM, and monitors privileged instruction simulation, interruption simulation, and so forth.

(2) Hardware monitor: This is built in a real host computer, and monitors CPU service time, supervisor state CPU service time, CPU wait time, NITLBR, Not in Buffer Ratio (NIBR), instruction execution counts, and so forth.

(3) Data analyzing program: This program sums up and analyzes the data gathered by both types of monitors.

5.3 Benchmark Jobs

Table 4 shows benchmark jobs. These jobs are Database Management Programs under a Hitachi OS with multiple virtual storage. They are of low CPU utilization and high I/O frequency. They are executed sequentially.

5.4 Performance Data

The following effects are expected by the performance improvement methods described in Sec. 4.

(1) Optimal number increase of the 2-0-translation tables,

(2) NITLBR, and MIET reduction,

(3) Total CPU overhead reduction.

These measurement data are discussed here as follows.

5.4.1 Optimal Number Increase of the 2-0-Translation tables

As for the number of 2-0-translation tables, it is enough to provide a VM with K 2-0-translation tables. Here, K equals the number of active level 2 virtual storage. However, a conventional VMS has disadvantages due to a full invalidation of 2-0-translation tables for PPTLB-simulation. Therefore, the total CPU overhead will be minimized at a smaller number of 2-0-translation tables than K. Fig. 5 shows this condition.

Namely, it reveals that the optimal number of 2-0-translation tables (N_0) equals 3 in the conventional VMS, though $K=6\sim7$ (That is, OS's master storage, Data Base management control storage, 5 job storage). On the contrary, in the VMS with MSA support, $N_0=6$, which is close to the value of K. Thus, the VMS with MSA support has the optimal number of 2-0-translation tables close to the number of active level 2 virtual storage. This means that MSA has sufficiently decreased the disadvantages of Multiple 2-0-Translation Table method.

5.4.2 Mean Instruction Execution Time Reduction

Fig. 6 shows the graphs of MIET, NITLBR and NIBR for the benchmark jobs. Each value is a relative value when that of the conventional VMS is set equal to 1. The number of 2-0-translation tables is set to the optimal number for each case, namely, 3 in the conventional VMS or 6 in the VMS with MSA support.

The figure shows that MSA has significantly reduced NITLBR to $1/2\sim1/4$. Moreover, it causes about a 10% reduction in the MIET. Thus, the effects of selective TLB purge is clear.

5.4.3 Total CPU Overhead Reduction

Fig. 7 illustrates the total CPU overhead reduction for the benchmark jobs. The number of 2-0-translation tables is set equal to the optimal number for each case.

Values for this figure are obtained by the following means.

(1) VMM CPU service time is the supervisor CPU service time measured by a hardware monitor.

(2) VMM items are calculated from the privileged instruction simulation counts measured by a software monitor.

(3) VM-Assist overhead is also calculated from the privileged instruction emulation counts.

This figure shows that 2-0-translation table maintenance overhead (i.e. VM-Assist's 2-0-translation table validation ①, VMM's PPTLB, Load Control (CR1) i.e. (LCTL), and Purge TLB simulation ③) forms $15\%\sim60\%$ of the total CPU overhead in the conventional VMS. Moreover, the 2-0-translation table maintenance overhead is almost entirely reduced with MSA, which causes about a $40\%\sim60\%$ reduction in the total CPU overhead.

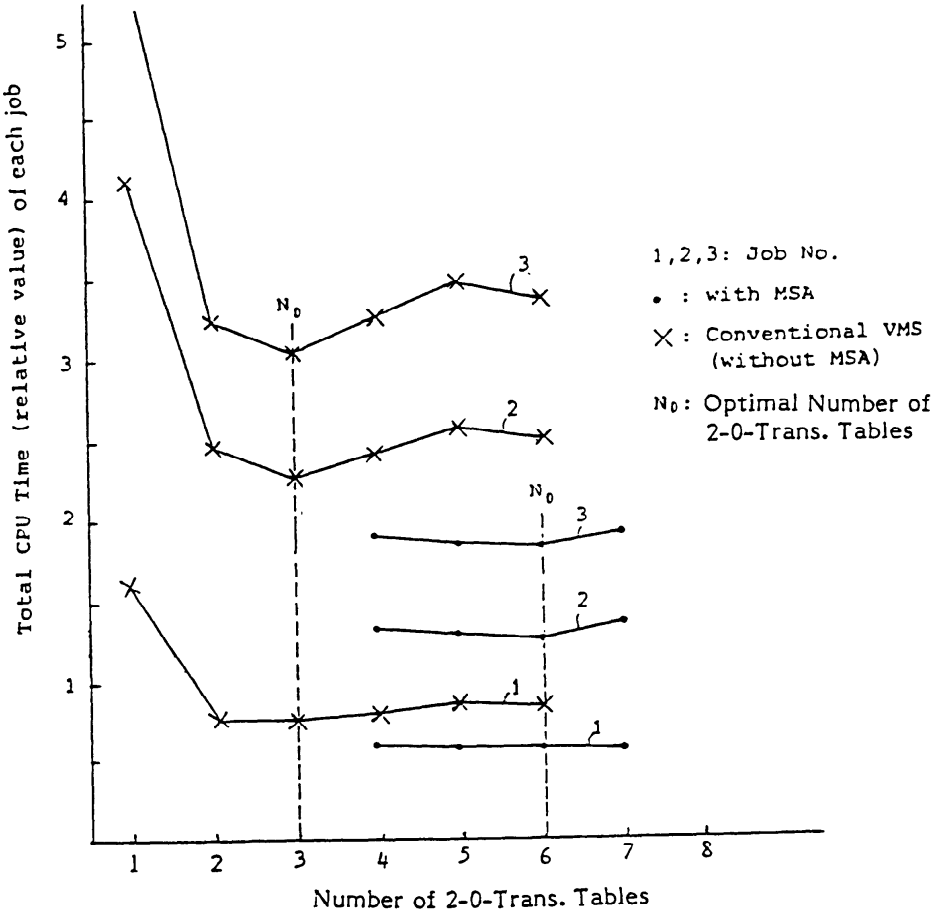
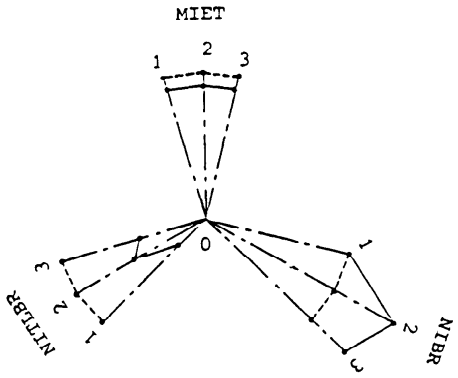


Fig. 5 Optimum Number of 2-0-Trans. Tables.



1, 2, 3 JOB NO.

——— With MSA

------- Conventional VMS (without MSA)

Fig. 6 Improvement of MIET & NITLBR.

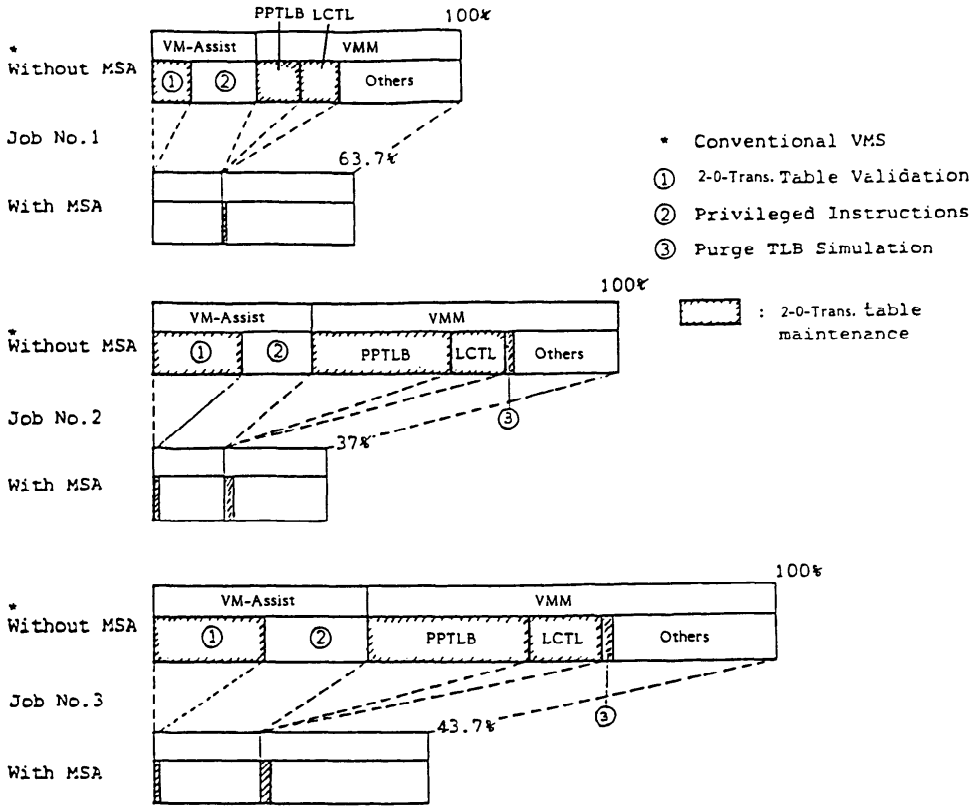


Fig. 7 Total CPU Overhead Reduction with MSA.

6. Conclusions

The Multi-Shadow Assist (MSA) implementing the fast 2-0-translation table switching and the selective invalidation of the 2-0-translation tables has been proposed. The MSA is applicable to any kind of VM memory attributes, and its effects do not depend upon them. The acceleration ratios of MSA are 3.4 ~ 24.4 when the number of 2-0-translation tables equals 3 ~ 6.

The effects of MSA have been confirmed by measurement for some benchmark jobs. The 2-0-translation table maintenance overhead is significantly reduced with MSA, which causes about a 10% reduction in the mean instruction execution time, and about a 40 ~ 60% reduction in the total CPU overhead.

Moreover, a prevention method of 2-0-translation table double invalidation has also been proposed for the temporary increase of the number of the OS's active virtual storage.

This performance improvement efficiently supports an OS with multiple virtual storage running under VMS using conventional hardware architecture.

References

1. GOLDBERG, R. P. Architectural Principles for Virtual Computer System, PH. D. Thesis, Div. of Engineering and Applied Physics, Harvard University, Cambridge, Ma., 1972.
2. GOLDBERG, R. P. Survey of Virtual Machine Research, *Computer* 7, 6 (June 1974), 34-35.
3. IMURA, J., KODAIRA, M., WAKAI, K. and UMENO, H. Kasou Keisanki, *Hitachi Hyouron* **61**, 12 (1979) (in Japanese).
4. HITAC, Introduction to VMS, 8040-3-001-30. HITAC Manual (in Japanese).
5. TAGUCHI, T., HORIKOSHI, H. and KURIHARA, J. Design and Experiments of a Virtual Machine System, *Information Processing Society of Japan* **20**, 4 (July 1979).
6. UMENO, H., OHMACHI, K., HINO, A. and IMURA, J. Development of a High Performance Virtual Machine System and Performance Measurements for It, *Journal of Information Processing* **4**, 2 (July 1981).
7. Hitachi Computer HITAC, HITAC Processor, 8080-2-001 (E).

(Received May 21, 1984; revised January 8, 1985)