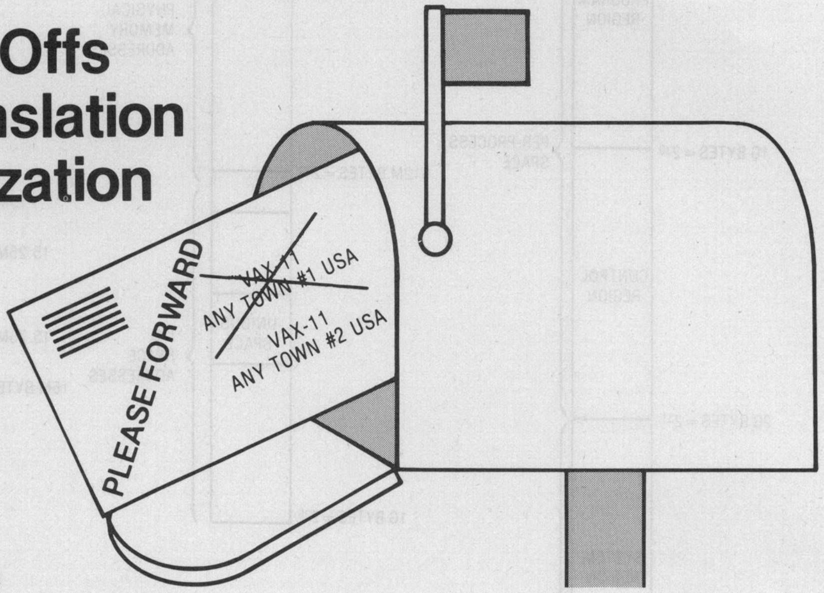
A major feature of the VAX-11 is its large virtual address space. This trace-driven simulation scheme evaluates address translation hardware that supports this feature cost-effectively.

Design Trade-Offs in VAX-11 Translation Buffer Organization



M. Satyanarayanan
Carnegie-Mellon University

Dileep Bhandarkar
Digital Equipment Corporation

One of the major architectural limitations of the PDP-11 has been its small virtual address space. This has been especially true during the last few years, as the level of sophistication of traditional minicomputer applications has risen. Consequently, the extension of virtual address space was a primary goal of VAX architects. (Overviews of VAX architecture are given by Strecker[1] and Levy and Eckhouse.[2]) A full 32-bit virtual address was chosen for VAX to ensure a long lifetime. Such a large address space requires an address translation mechanism other than that used by the 64K-byte virtual address space of the PDP-11.

The PDP-11 uses page tables that are architecturally specified as processor registers, which must be updated by the operating system at context switch time. Such a scheme is impractical for a large address space. The PDP-11 needs eight page-table entries per processor mode to map eight pages, each of which is up to 8K bytes long. Even with an 8K-byte page size, it would take 512K page-table entries to map the 4G bytes (G = 2 ** 30) of address space provided by VAX. The VAX architecture uses a 512-byte page to reduce internal fragmentation and increase effective physical memory utilization.

Detailed justification of the design trade-offs that led to the memory management architecture are beyond the present scope. This article deals with the analysis of the design trade-offs for the hardware implementation of the VAX address translation mechanism.

Overview of VAX memory management

The four-gigabyte address space is divided into four regions, as shown in Figure 1. The first two regions, *program* and *control*, comprise the per-process virtual address space, which is uniquely mapped for each process. The other two regions make up the *system virtual address space* that is common to all processes. The program region contains user programs and data. The control region contains the user stack and operating system data structures specific to the user process. The system region contains procedures and data common to all user processes, and page tables.

Virtual and physical address formats are shown in Figure 2. Bits 8:0 specify a byte within a 512-byte page as the basic mapping entity. Bits 29:9 specify a virtual page number, or VPN, within one of the four virtual address regions. The system region has its page table defined in physical memory by a base-and-length register. Thus, the system-region page table is contiguous in physical memory. The per-process space page tables are defined by the program and control region base-and-length registers. However, these base registers do not contain physical addresses; rather, they contain system-region virtual addresses. Thus, the per-process page tables are contiguous in the system-region virtual address space and are not necessarily contiguous in physical memory. This allows the per-process page tables to be paged.

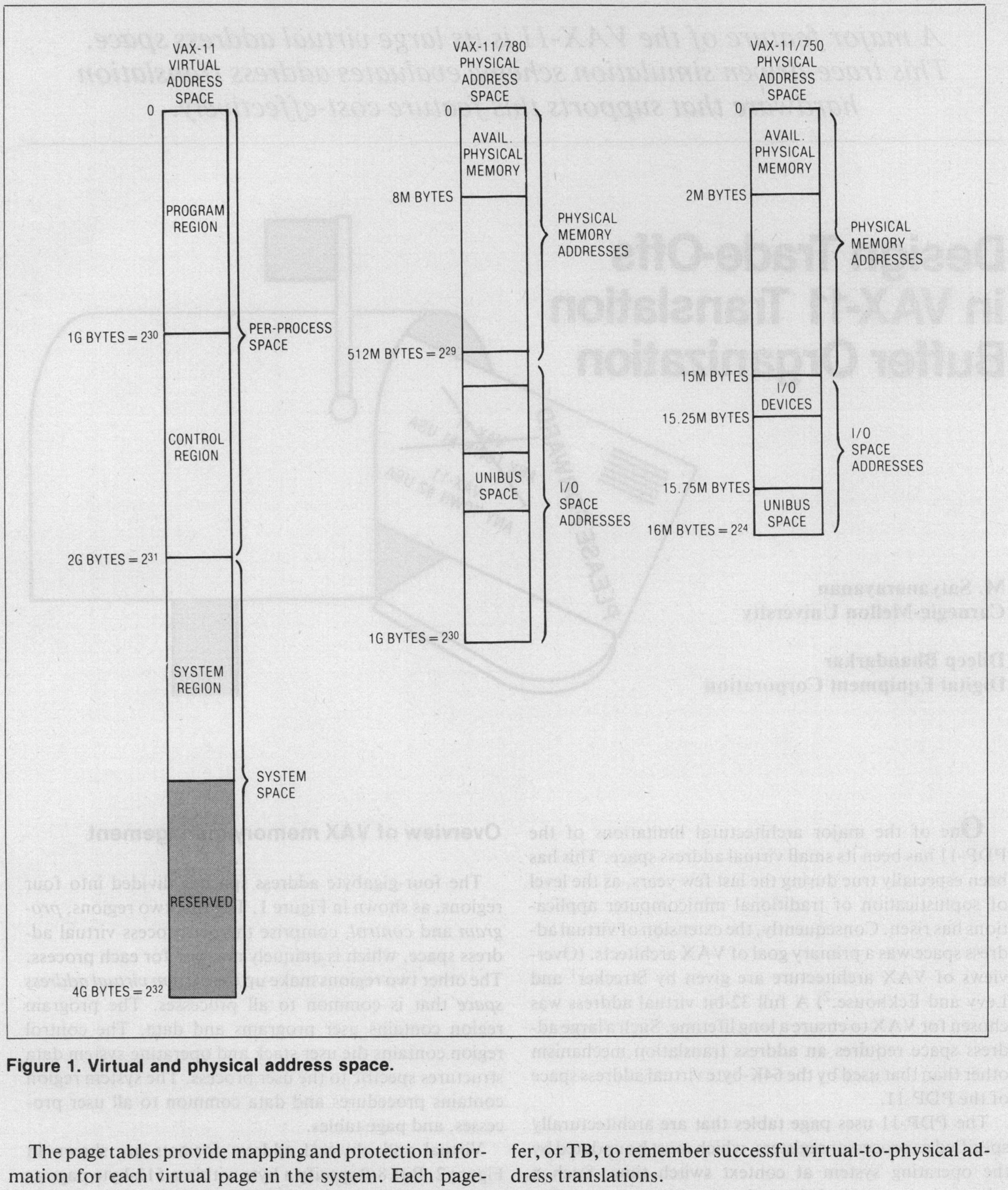


Figure 1. Virtual and physical address space.

The page tables provide mapping and protection information for each virtual page in the system. Each page-table entry is 32 bits long, as shown in Figure 3. To translate a virtual address to a physical address, the processor simply uses the VPN as an index into the appropriate page table, which was taken from the given page-table address. Note that the per-process virtual address requires two memory accesses to generate a physical address: one to access the system page table, to determine the physical address of the page-table entry, or PTE; and one to access the PTE itself. Such overhead would be unacceptable if it occurred on every memory access. In order to save actual memory references when repeatedly referencing the same set of pages, VAX implementations include a hardware mechanism called a translation buffer, or TB, to remember successful virtual-to-physical address translations.

Translation buffer overview

The translation buffer can be viewed as a cache for address translation. Both the VAX-11/780 and the VAX-11/750 contain a two-way set-associative TB. The VAX-11/780 TB can cache up to 128 address translations; the VAX-11/750 TB has up to 512 entries. In both machines, the TB is divided into halves: one for per-process regions, and the other for the system region. The per-process TB entries are automatically invalidated by the processor when a *load process context* instruction is

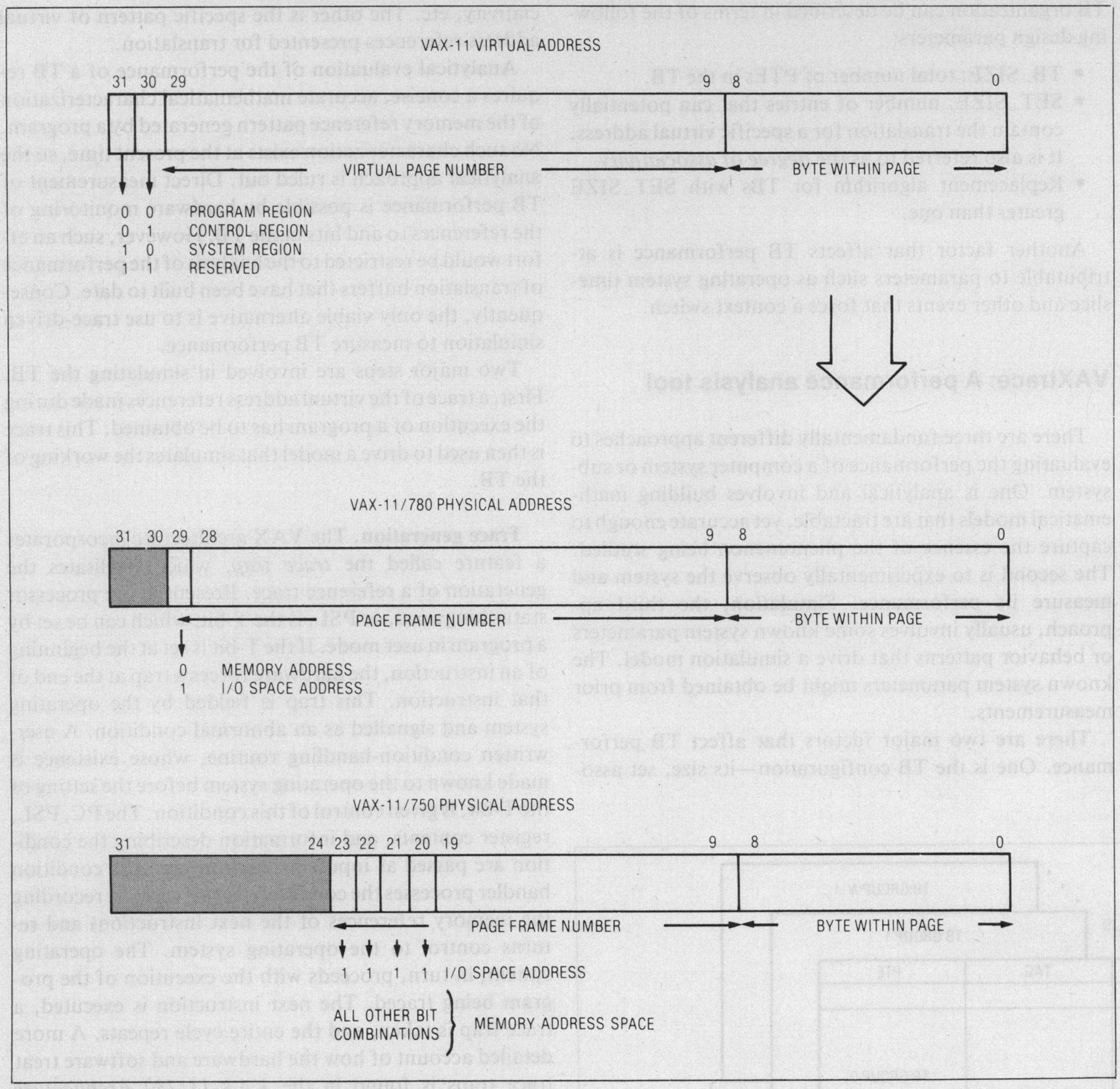


Figure 2. Virtual and physical address formats.

executed. The architecture allows the operating system to selectively invalidate all entries or an entry corresponding to a particular virtual address. Such invalidation is necessary when a PTE is changed in the page tables by the operating system.

A canonical diagram of a TB is shown in Figure 4. The TB consists of several identical sets. The number of such sets is termed SET_SIZE. Each set contains a tag field and a physical page frame number, or PPFN, field. The index part of the virtual address is used to index into one entry in each set. If the tag field of the virtual address matches the tag stored in the TB entry, a hit occurs and the PPFN is used to generate the physical address. Figure 5 shows how the virtual address is used by the TB in the VAX-11/780.[3]

The performance of the TB can be measured in terms of its hit ratio, i.e., the fraction of translations found in the TB. (We chose a slightly different parameter, for reasons to be explained later.) The hit ratio is a function of both the TB organization and the program characteristics. The

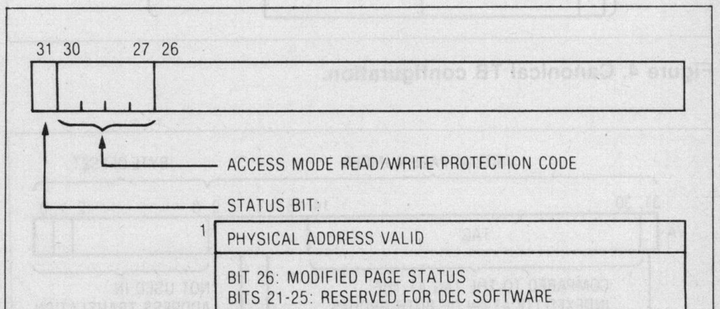

31 30
27 26
ACCESS MODE READ/WRITE PROTECTION CODE
STATUS BIT:
1
PHYSICAL ADDRESS VALID
BIT 26: MODIFIED PAGE STATUS
BITS 21-25: RESERVED FOR DEC SOFTWARE
BITS 0-20: PAGE FRAME NUMBER
0
ACTIVATE PAGING SOFTWARE ON REFERENCE (PAGE FAULT)
BITS 0-26 MAY INDICATE:
DISK SECTOR IDENTIFICATION
TEMPORARY POINTER TO SHARED PAGE

Figure 3. Page table entry.

TB organization can be described in terms of the following design parameters:

- TB_SIZE: total number of PTEs in the TB.
- SET_SIZE: number of entries that can potentially contain the translation for a specific virtual address. It is also referred to as the *degree of associativity.*
- Replacement algorithm for TBs with SET_SIZE greater than one.

Another factor that affects TB performance is attributable to parameters such as operating system time-slice and other events that force a context switch.

VAXtrace: A performance analysis tool

There are three fundamentally different approaches to evaluating the performance of a computer system or subsystem. One is analytical and involves building mathematical models that are tractable, yet accurate enough to capture the essence of the phenomenon being studied. The second is to experimentally observe the system and measure its performance. Simulation, the third approach, usually involves some known system parameters or behavior patterns that drive a simulation model. The known system parameters might be obtained from prior measurements.

There are two major factors that affect TB performance. One is the TB configuration—its size, set associativity, etc. The other is the specific pattern of virtual address references presented for translation.

Analytical evaluation of the performance of a TB requires a concise, accurate mathematical characterization of the memory reference pattern generated by a program. No such characterization exists at the present time, so the analytical approach is ruled out. Direct measurement of TB performance is possible by hardware monitoring of the references to and hits in the TB. However, such an effort would be restricted to the analysis of the performance of translation buffers that have been built to date. Consequently, the only viable alternative is to use trace-driven simulation to measure TB performance.

Two major steps are involved in simulating the TB. First, a trace of the virtual address references made during the execution of a program has to be obtained. This trace is then used to drive a model that simulates the working of the TB.

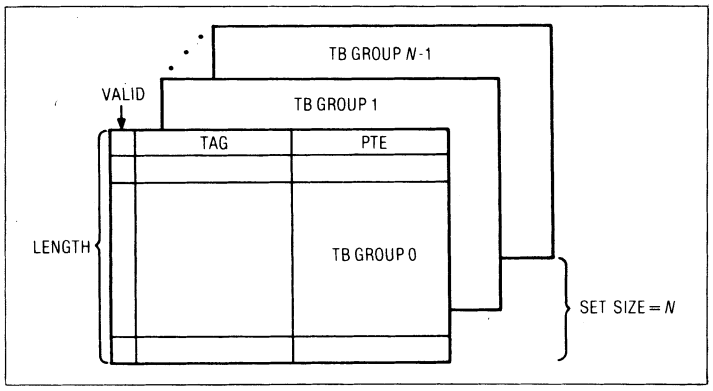


Figure 4. Canonical TB configuration.

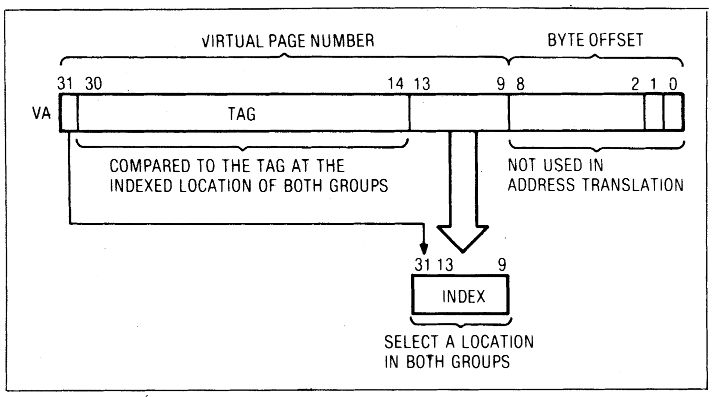

VIRTUAL PAGE NUMBER
BYTE OFFSET
31 30
14 13
9 8
2 1 0
VA
TAG
COMPARED TO THE TAG AT THE INDEXED LOCATION OF BOTH GROUPS
NOT USED IN ADDRESS TRANSLATION
31 13
9
INDEX
SELECT A LOCATION IN BOTH GROUPS

Figure 5. TB usage of virtual address fields.

Trace generation. The VAX architecture incorporates a feature called the *trace trap,* which facilitates the generation of a reference trace. Present in the processor status longword, or PSL, is the T-bit, which can be set by a program in user mode. If the T-bit is set at the beginning of an instruction, the hardware forces a trap at the end of that instruction. This trap is fielded by the operating system and signalled as an abnormal condition. A user-written condition-handling routine, whose existence is made known to the operating system before the setting of the T-bit, is given control of this condition. The PC, PSL, register contents, and information describing the condition are passed as inputs to this routine. The condition handler processes the condition (in this case, by recording the memory references of the next instruction) and returns control to the operating system. The operating system, in turn, proceeds with the execution of the program being traced. The next instruction is executed, a trace trap is taken, and the entire cycle repeats. A more detailed account of how the hardware and software treat trace traps is found in the *VAX-11/780 Architecture Handbook.*[4]

The user-written condition-handling routine is linked to the program to be traced. The resulting executable module is then run. This run creates a trace file that contains every virtual address reference made by the program in user mode. This file uses a simple data compression scheme to reduce the physical length of the trace. It contains information on the type of each reference (read, write, or modify) and miscellaneous information such as whether the reference is to the instruction stream, whether there is a break in the sequential flow of instruction stream references, etc. The trace file is used as input to the TB simulation program.

TB simulation program. The program used to simulate the TB is relatively straightforward. Originally, the entire program was written in Fortran, but (after some use) certain parts of it were recoded in assembly language to improve performance. The TB itself is divided into two multidimensional arrays, with the number of dimensions being equal to the number of sets in the TB. One array represents the validity bits of the TB, and the other represents the tags (the virtual addresses corresponding to each of the valid entries). The physical address mapping information is not maintained, since no translation actually occurs; only look-ups and updates of the tag and validity portions of the TB take place.

For TBs with a set size greater than one, a random selection rule is used to select the TB group (see Figure 4) whose entry is to be replaced. We did not investigate the effect of replacement algorithms; similar studies of program cache performance have shown the replacement algorithm to be of secondary importance.[5]

The VAX-11/780 has a pretranslation mechanism as part of its instruction buffer (I-buffer). This mechanism keeps track of the physical page corresponding to the current instruction stream (I-stream) virtual page. References to the I-stream can be translated by the I-buffer without using the TB or page tables, as long as the instruction flow is sequential and a page boundary is not crossed. Pretranslation is used in the simulation program, too; however, it can be turned off in order to model a machine like the VAX-11/750, which does not use pretranslation.

Context switching is usually detrimental to TB performance. When a context switch occurs in the VAX, the mapping tables for the per-process part of the TB must be invalidated. If context switches occur often enough, the frequent clearing of the TB can cause significant performance degradation. The simulation program incorporates code to simulate the periodic clearing of the TB. The time between clearings is assumed to be exponentially distributed; the user can specify the mean of this distribution.

Measurements

The function of a TB is to speed up the translation process by reducing the average amount of time needed to perform a PTE look-up. Let p be the probability of finding the desired PTE in the TB. Further, let $t1$ be the access time for a PTE when it is in the TB; let $t2$ ($>t1$) be the access time when it is not. The expected time for performing a page-table look-up is $p * t1 + (1-p) * t2$. The quantities $t1$ and $t2$ are functions of the specific hardware implementation, a factor that includes clock rates, bus speeds, memory speed, and logic implementation. The quantity p is dependent on both the TB configuration and the specific program being observed. A change in any TB parameter usually affects p significantly. Consequently, p is often used as a measure of TB performance and is called the *hit ratio*. The quantity $1-p$ is an equally valid performance measure; it is called the *miss ratio*.

As described earlier, a reference to a virtual address in process space might need two levels of translation before it can be mapped into the corresponding physical address. The first of these translations is for an address generated by the program. This address is called an *explicit reference*. The second is for an implicit address generated in the translation process. Using the hit ratio as a measure of performance is not quite appropriate here, since it would ignore implicit references. The single number that best characterizes VAX TB performance is the average number of *main memory references to page tables* per *explicit virtual memory reference*. If each explicit virtual

we're SINGLE and . . . we're AVAILABLE

BACK ISSUES

of . . .
COMPUTER, IEEE Computer Graphics and Applications, IEEE MICRO.

COMPLETE YOUR LIBRARY TODAY. USE ORDER FORM AT BACK OF BOOK.

ORDER NOW. Prepaid orders only.
Supplies of hard copies are limited.
See chart below for issues available in microfiche only.

		Prepaid
COMPUTER	members	$3.00
	nonmembers	$6.00
GRAPHICS & MICRO	members	$6.00
	nonmembers	$12.00

COMPUTER

	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81
JAN	M			M	M	M			M		M	M	M		M
FEB	M	M	M	M	M	M			M		M				
MAR	M	M		M					M			M	M		
APR	M	M	M	M	M	M				M	M				
MAY		M	M				M		M						
JUN	M	M	M	M	M	M									
JUL	M	M	M	M											
AUG	M	M			M	M					M				
SEPT															
OCT	M	M	M	M	M	M									
NOV															
DEC	M	M	M	M	M	M						M			

M = microfiche only.

memory reference results in one TB lookup (as is the case in most other virtual memory systems), this number would equal the miss ratio. In the present case, implicit references cause this number to be slightly higher than the miss ratio. We use the average number of memory references per translation as the figure of merit for a TB configuration.

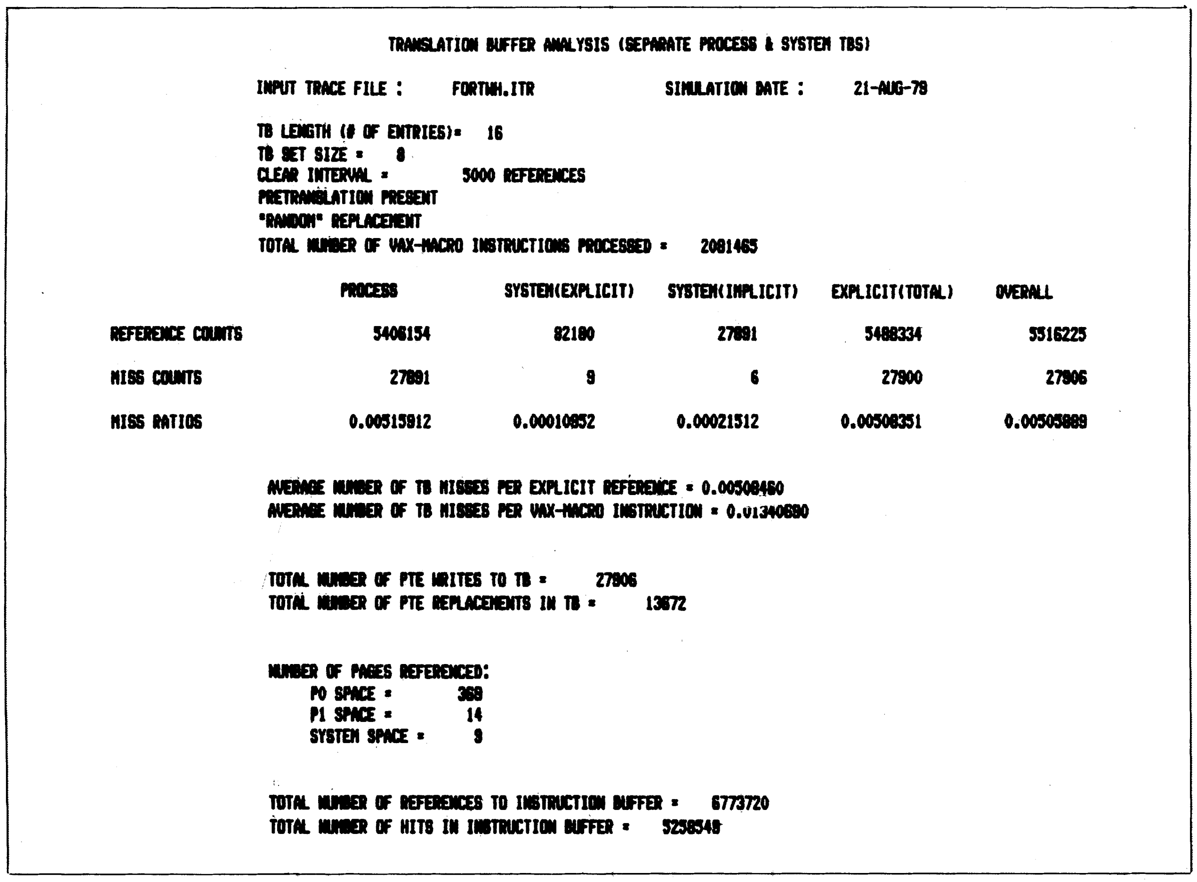
Because of the presence of pretranslation in the VAX-11/780, many references to the I-stream are not subjected to the usual translation process. Consequently, the total number of references to the TB is less than the number of references generated by the program. In computing the figure of merit of the TB, we count only those program-generated references that are actually presented to the TB as explicit.

Raw results. Figure 6 is an example of the data generated by the TB simulation program. The first few lines identify the trace file and the program whose execution was traced. Below this are the parameters of the particular TB configuration. TB_SIZE, the total number of elements in the TB, is the product of TB_LENGTH and SET_SIZE. The clear interval is the mean of the exponential distribution used to simulate the periodic clearing of the TB.

The simulation results follow these preliminary details. References are classified into process space references, explicit system space references, and implicit system space references. The fourth column lists the total number of explicit references (system + process). The final column lists all references—explicit and implicit, system and process. The number of references of each of these types, the number of misses in each case, and the miss ratios follow. The ratio of the overall number of misses to the total number of explicit references gives the average number of main memory references per explicit translation. The number of references to the I-stream and the number of references for which translation was avoided because of pretranslation in the I-buffer are found in the last two lines.

Interpretation of the data. In this study, four quantities that could affect TB performance were varied and the effects observed. These quantities are the TB length, TB set size, the clear interval, and the traced program itself. Figures 7 through 10 depict the effect of each of these variables on TB performance. Again, the number of memory references per explicit translation is the measure of performance. Since the trace program only collects references in user mode, and since there are very few system space references in user mode, this quantity is very close to the miss ratio in most virtual memory systems.

The performance measure excludes address translations not presented to the TB due to pretranslation hits. Typically, about two-thirds of all translations can be avoided by pretranslation. Without pretranslation, most



```
                      TRANSLATION BUFFER ANALYSIS (SEPARATE PROCESS & SYSTEM TBS)

             INPUT TRACE FILE :      FORTWH.ITR                 SIMULATION DATE :      21-AUG-79

             TB LENGTH (# OF ENTRIES)=   16
             TB SET SIZE =    8
             CLEAR INTERVAL =        5000 REFERENCES
             PRETRANSLATION PRESENT
             "RANDOM" REPLACEMENT
             TOTAL NUMBER OF VAX-MACRO INSTRUCTIONS PROCESSED =    2081465

                          PROCESS          SYSTEM(EXPLICIT)   SYSTEM(IMPLICIT)   EXPLICIT(TOTAL)    OVERALL

REFERENCE COUNTS             5406154             82180              27891            5488334        5516225

MISS COUNTS                    27891                 9                  6              27900          27906

MISS RATIOS               0.00515912        0.00010852         0.00021512         0.00508351     0.00505889


              AVERAGE NUMBER OF TB MISSES PER EXPLICIT REFERENCE = 0.00508460
              AVERAGE NUMBER OF TB MISSES PER VAX-MACRO INSTRUCTION = 0.01340680


              TOTAL NUMBER OF PTE WRITES TO TB =     27906
              TOTAL NUMBER OF PTE REPLACEMENTS IN TB =     13672


              NUMBER OF PAGES REFERENCED:
                  P0 SPACE =       369
                  P1 SPACE =        14
                  SYSTEM SPACE =     9


              TOTAL NUMBER OF REFERENCES TO INSTRUCTION BUFFER =    6773720
              TOTAL NUMBER OF HITS IN INSTRUCTION BUFFER =    5258548
```

Figure 6. Data generated by the TB simulation program

of these sequential instruction stream references would result in TB hits. Pretranslation is faster than a TB hit, but requires a little extra hardware.

Figure 7 shows the effect of varying the TB length. Performance improves as length increases, because more PTEs can be held in the TB at any given time. If the working set of a program always consists of contiguous virtual pages, increasing the TB length will always improve TB performance. The reason for this is that as the TB length increases, the probability that two pages in the working set will map into the same TB slot decreases.

In practice, the working sets of typical programs do not consist of contiguous pages, so there is an inevitable flattening of the performance curve as the TB length increases. The point at which the flattening sets in depends on both the program and the other TB variables. Figure 6 indicates that for a set size of one, increasing the TB length up to 256 increases performance significantly. For set sizes of two and four, increasing the length beyond 128 and 64, respectively, is of only marginal value. (Note that both the VAX-11/780 and the more recent VAX-11/750 use a set size of two.) The capacity of the fast-memory chips available in the mid 1970's limited the length of the VAX-11/780 TB to 64. The VAX-11/750, which was introduced three years later, uses a chip with four times as many bits. The TB length for the VAX-11/750 is 256.

The effect of set size on performance is shown in Figure 8. Each of the curves is a constant TB_SIZE curve; i.e., the total number of entries in the TB is constant. For instance, a TB of size 64 can be configured with a set size of one and a length of 64, or a set size of two and a length of 32, or a set size of four and a length of 16, and so on. The limiting case is a TB with a set size equal to the number of entries; i.e. a fully associative TB. For a given TB_SIZE, as the set size increases, so does the number of entries into which a given virtual address can map. Consequently, even if two pages in the working set have the same index, their page-table entries can coexist in separate sets, improving TB performance. For a given TB_SIZE, the best performance is obtained at the maximum set size. Figure 8 shows that for the program being studied, increasing the set size from one to two and from two to four improves performance significantly. Beyond this, the program characteristics are such that increasing the set size does not pay off. Our studies of other programs indicate that in most cases significant improvement occurs up to a set size of four.

We studied the effects of three programs: a Fortran compilation, a Macro assembly, and a link. Figure 9 illustrates the effects of program behavior. One obvious conclusion is that there can be no "typical" program; the performance of the TB depends significantly on the program being observed. Still, the general shapes of the curves are the same, and this indicates that varying the TB configuration has the same qualitative effect on the performance of different programs. The actual numbers one obtains, however, depend on the program itself.

The graph shows an apparent anomaly for the curves corresponding to a set size of one. A Macro assembly requires more memory references than a Fortran compilation for TB lengths of 32 and 64; increasing the length to

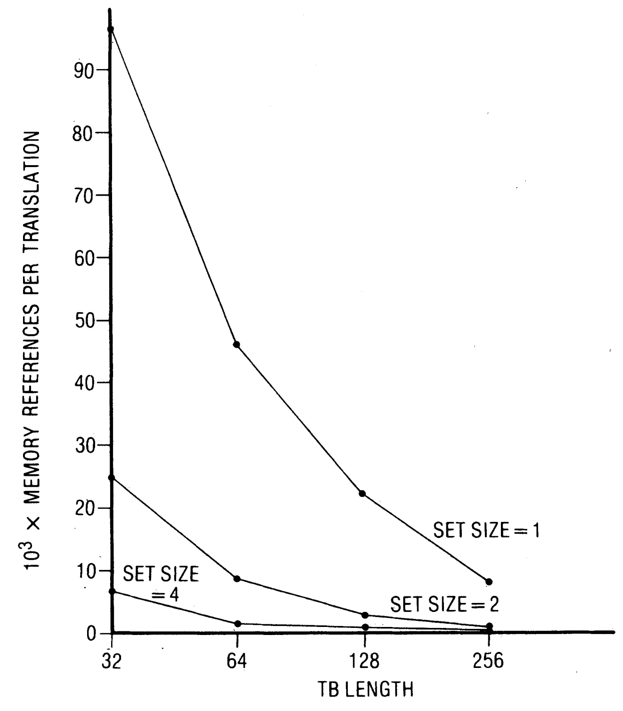


Figure 7. Effect of TB length—Fortran compilation, pretranslation present.

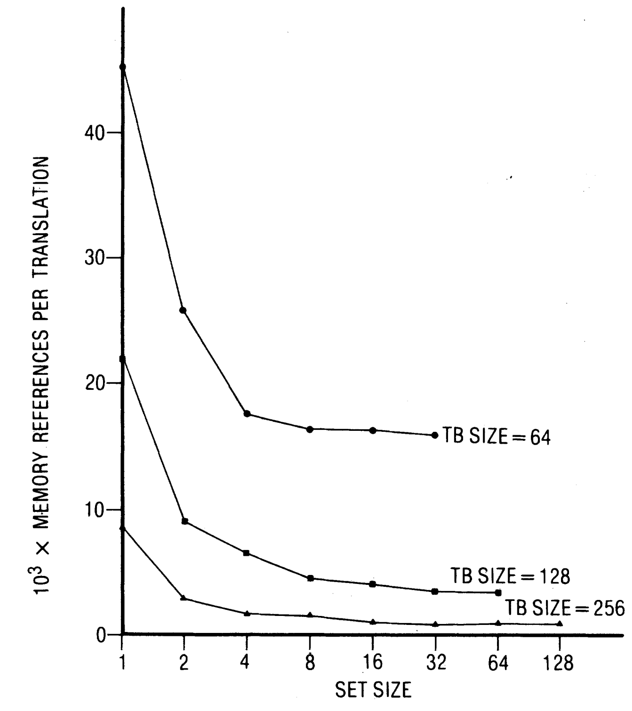


Figure 8. Effect of set size—Fortran compilation, pretranslation present.

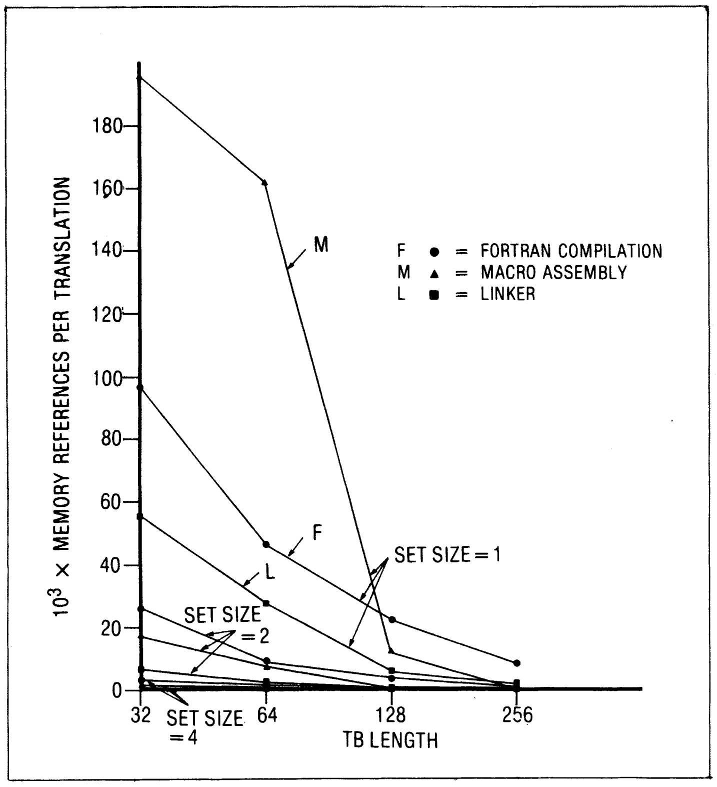


Figure 9. Effect of program variation.

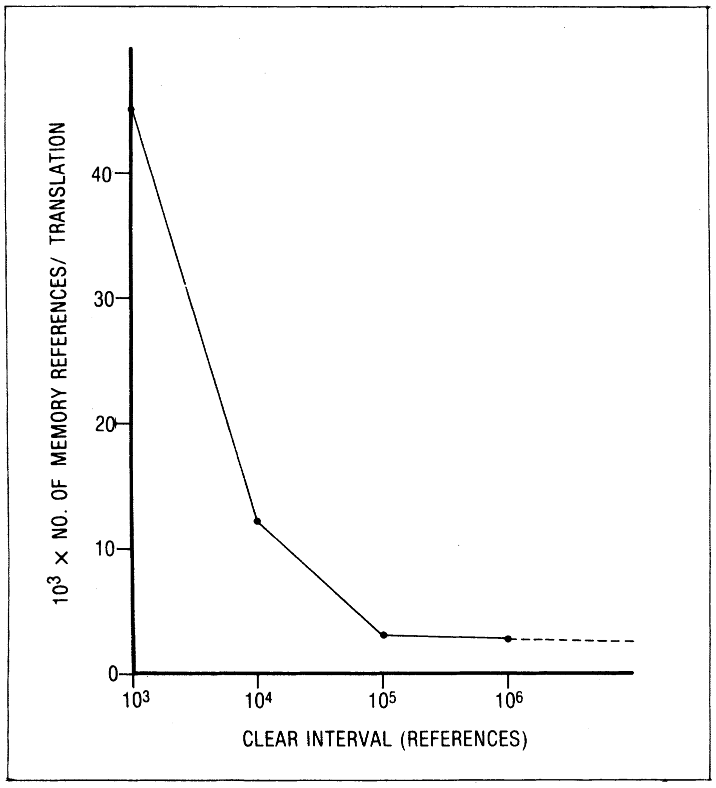


Figure 10. Effect of clearing TB—Fortran compilation: TB length = 128, set size = 2.

128, however, yields more memory references for Fortran compilation than for Macro assembly. Such variations in TB performance should not be considered anomalies, but manifestations of differences in the behavior of different programs.

Figure 10 shows what happens to performance as the user portion of the TB is cleared at periodic intervals. Each clearing usually corresponds to a context switch in a multiprogramming environment. Shortening the clear interval to 1000 references degrades performance significantly. There is a noticeable loss in performance even at a clear interval of 10,000 references. Larger clear intervals yield performance figures close to the asymptotic value obtained when the TB is never cleared. Our measurements of typical educational timesharing work loads indicate that the clear interval is around 25,000 references.

Further investigations. The current measurement work can be extended in several directions. The effect of replacement algorithms needs to be analyzed; our suspicion that it is a second-order effect needs to be validated. Traces for system routines could also be analyzed. Our study analyzed TBs with equal sections reserved for per-process and system virtual addresses. Generation of a more composite trace that includes the operating system references would open several other options to analysis. One alternative is to use a single large TB, all of which is flushed at context switch time. Another is to configure the two separate sections independently. It would also be interesting to integrate this model of the TB into an overall system model to investigate the overall translation time as a function of not only the TB configuration, but factors such as instruction buffer speed and length, cache configuration, memory system implementation, and processor organization.[6] The behavior of very large programs (much greater than a megabyte) also remains to be analyzed.

Conclusions

The data suggests certain broad conclusions about TB design:

(1) The specific program is as important a factor in determining performance as any of the design parameters of the TB. To predict the performance of a specific TB design, one must observe its performance on a variety of the type of programs that will make up most of the work load of the computer being designed. If a single number is needed to characterize the performance of a TB design, these individual performance measures must be weighted in some way, perhaps by the relative frequency of use of the programs to which they correspond. However, the different programs we analyzed exhibited the same qualitative behavior. All reached a point of diminishing marginal performance improvement at approximately the same TB configuration.

(2) Set associativity is very desirable. Significant performance improvement is seen in enlarging set size from one to two. Beyond a set size of four, performance improvement is marginal.

(3) The frequency of context switches can affect TB performance significantly.

Measurements must always be evaluated in conjunction with other implementation factors. One cannot, for instance, conclude that a set size of two is always the best choice. Cost effectiveness might be better for a 256×1 configuration than for a 64×2. Note that cost is a function of the actual configuration, not merely the size, and that performance requirements for the TB are likely to vary for different implementations. The overall translation time is also affected by the memory and processor cycle times.

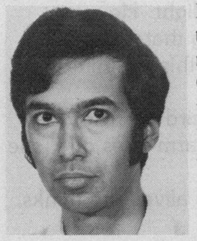
A great many decisions must be made in this area. The quantitative information presented here should help the hardware implementer make intelligent design tradeoffs. ■

References

1. W. D. Strecker, "VAX-11/780—A Virtual Address Extension to the DEC PDP-11 Family," *AFIPS Conf. Proc.*, 1978 NCC, June 1978.
2. H. M. Levy and R. H. Eckhouse, Jr., *Computer Programming and Architecture: The VAX-11*, Digital Press, Bedford, Mass., 1980.
3. *VAX-11/780 TB/Cache/SBI Technical Description*, Digital Equipment Corp., Maynard, Mass., 1978.
4. *VAX-11/780 Architecture Handbook*, Digital Equipment Corp., Maynard, Mass., 1980, pp. 305-308.
5. W. D. Strecker, "Cache Memories for the PDP-11 Family Computers," *Proc. Third Ann. Symp. Computer Architecture*, Jan. 1976, pp. 155-158.
6. C. A. Wiecek and S. C. Steely, "Performance Simulation as a Tool in Central Processor Unit Design," *Proc. Conf. Simulation, Measurement, and Modeling*, Aug. 1979, pp. 41-47.

M. Satyanarayanan is currently working toward a PhD in computer science at Carnegie-Mellon University. He holds a B. Tech in electronics and an M. Tech in computer science, both from the Indian Institute of Technology, Madras. His interests include computer architecture and operating systems, particularly those of multiprocessor systems, and the design and evaluation of memory hierarchies. He is the author of the book *Multiprocessors: A Comparative Study*.

Satyanarayanan has worked for IBM and DEC, and is a member of the IEEE Computer Society and the ACM.



Dileep P. Bhandarkar is the manager of the VAX/PDP-11 Architecture Management Department at Digital Equipment Corporation. Earlier, he was a member of the technical staff in corporate development at Texas Instruments. He holds PhD and MS degrees in electrical engineering from Carnegie-Mellon University, and a B. Tech in electrical engineering from the Indian Institute of Technology, Bombay.

A member of the IEEE Computer Society, he holds US patents on magnetic-bubble memory organization and fault-tolerant monolithic memories.

INTERMETRICS

THE LEADER IN COMPILER TECHNOLOGY

INTERMETRICS, INC. is recognized as an international leader in the state-of-the-art design and implementation of multi-targeted, highly optimizing compilers integrated into complete software development systems. In addition, we are committed to the continued development of sophisticated tools as the keystone of powerful and supportive programming environments. We are seeking special individuals to perform in these highly visible, key positions. Openings exist in our Bethesda, Maryland and Cambridge, Massachusetts facilities.

COMPILER DESIGNERS

To assume lead technical roles in the design and implementation of compilers for such languages as FORTRAN, Ada, Pascal, Jovial and HAL.

SENIOR SOFTWARE TEST ENGINEERS

To direct the design and implementation of a test suite to validate new compilers' language coverage and implementation of translation and optimization algorithms.

SUPPORT SOFTWARE SPECIALISTS

To specify, design and implement editors, debuggers, linkers, loaders, and static analyzers in the development of user-friendly systems.

We offer an outstanding benefits program, flexible hours, and a results-oriented, informal environment which encourages a variety of recreational and intellectual activities.

For further information, call Karen Aylmer collect at **(617) 661-1840** or write to:

Personnel Department
INTERMETRICS, INC.
733 Concord Avenue
Cambridge, MA 02138

Permanent Residency or U.S. Citizenship Required
An Equal Opportunity Employer

INTERMETRICS