Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement

DOUGLAS W. CLARK AND JOEL S. EMER
Digital Equipment Corporation

A virtual-address translation buffer (TB) is a hardware cache of recently used virtual-to-physical address mappings. The authors present the results of a set of measurements and simulations of translation buffer performance in the VAX-11/780. Two different hardware monitors were attached to VAX-11/780 computers, and translation buffer behavior was measured. Measurements were made under normal time-sharing use and while running reproducible synthetic time-sharing work loads. Reported measurements include the miss ratios of data and instruction references, the rate of TB invalidations due to context switches, and the amount of time taken to service TB misses. Additional hardware measurements were made with half the TB disabled. Trace-driven simulations of several programs were also run; the traces captured system activity as well as user-mode execution. Several variants of the 11/780 TB structure were simulated.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles—*associative memories*; *cache memories*; *virtual memory*; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids—*simulation*; C.1.1 [**Processor Architectures**]: Single Data Stream Architectures—*VAX*

General Terms: Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Translation buffer, translation look-aside buffer, miss ratio, hardware monitor, trace-driven simulation

1. INTRODUCTION

Virtual-memory systems profit from the use of a fast hardware cache of virtual-to-physical address translations. This cache, here called a *translation buffer* (TB), is sometimes known as a Directory Look-Aside Table (DLAT) or a Translation Look-aside Buffer (TLB). Translation buffers are widely used, but their performance is rarely reported (see [17, 18]). In this paper we report the performance of the translation buffer used in the Digital Equipment Corporation VAX-11/780 computer [5, 20]. We use two methods: trace-driven simulation and direct measurement of the hardware.

Virtual memory requires a translation of each virtual address generated by a program into a physical address, which is used to reference real memory. Most

One author of this paper is an Associate Editor of Transactions on Computer Systems; he was not involved in any way in the editorial decision process that resulted in acceptance of this paper for publication.

Authors' addresses: D. W. Clark, Digital Equipment Corp., 295 Foster Street, Littleton, MA 01460; J. S. Emer, Digital Equipment Corp., 77 Reed Road, Hudson, MA 01749.

virtual-memory systems use two levels of mapping, in which an address is interpreted first as belonging to a *segment* and then translation is performed for the specified *page* within the segment. Determination of a segment address from a virtual address may be simple, as in the VAX or the IBM System/370 [2], or complex, as in the Intel iAPX 432 [10].

If the virtual memory is large, the tables required to hold the page translations are typically so big that they must be stored in main memory. Therefore, in general, each virtual-memory reference logically requires one or more extra memory references just to do the translation.

A translation buffer is a high-speed associative cache of recently used virtual-to-physical address translations. Its purpose is to eliminate the need for the extra memory references most of the time by taking advantage of the principle of *locality* [4]. When an address translation is present in the TB, main memory need not be read to perform the translation; instead the mapping is performed in hardware with the aid of the buffer. This event is called a translation buffer *hit*. When a desired translation is not in the buffer—a TB *miss*—the necessary table or tables must be referenced and the translation constructed and inserted into the TB, perhaps displacing a previous entry. The time lost servicing TB misses is one of the costs of supporting virtual memory.

Optimizing TB performance involves a cost/performance evaluation of three items: the TB miss rate, the time to perform a virtual-to-physical translation for a TB hit, and the time required to fill a TB entry following a miss. The miss rate is directly related to the TB configuration, where bigger and more associative is better. But cost constraints, timing limitations on the associative hardware, and practical consideration of the depth of available RAM memory chips lead to designs with limited size and associativity. The translation time for a TB hit is generally highly constrained by the machine cycle time and cache access requirements, which can also place restrictions on the feasibility of some configurations. Finally, the fill time is related to memory reference times and the implementation of the translation algorithm.

In the rest of this paper we study the consequences of these design trade-offs on performance of the VAX-11/780. After giving some necessary definitions in Section 2, we discuss in Section 3 the techniques we used to characterize the 11/780 TB. Section 4 presents the raw miss data as measured and simulated, and categorizes the sources of the misses. Section 5 reports measurements of the TB miss service time and illustrates its impact on overall CPU performance. Section 6 examines the frequency of some architectural events that affect TB performance, such as context switches and branches. Section 7 discusses performance of alternative TB configurations, and Section 8 concludes the paper.

2. DEFINITIONS

2.1 VAX Virtual Memory

VAX virtual memory consists of a 32-bit address space that is divided into three usable regions or segments. These regions are selected by the top two bits of the virtual address. They are referred to as the *system-space* region (S0), and two *process-space* regions (P0 and P1). Each region is in turn partitioned into pages

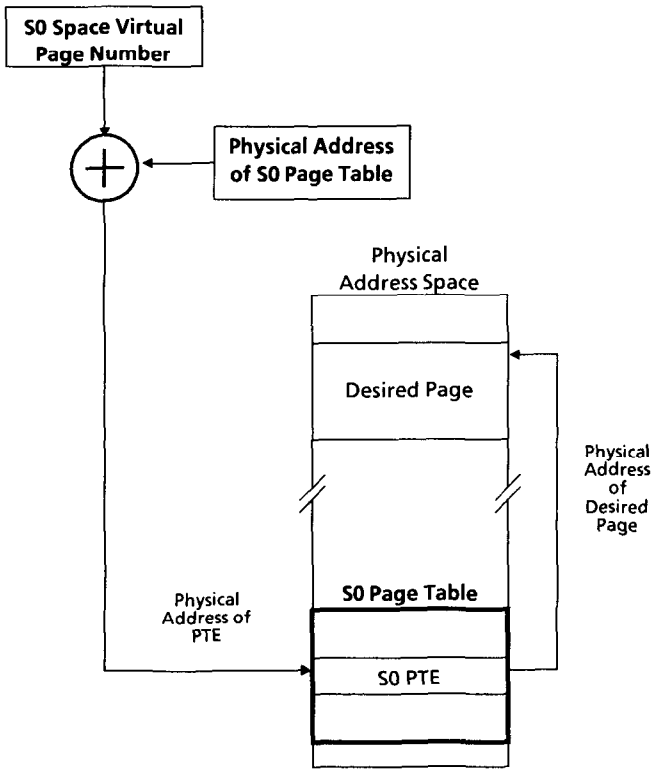

Fig. 1. System space virtual page translation.

of fixed size. Each page contains 512 bytes, and the low-order 9 bits of each address determine the byte within the page containing the desired reference.

Since virtual-to-physical address translation in the S0 region is simplest, we will consider it first. Given a virtual address, the translation for a given page can be found by looking at the proper *page table entry* (PTE) in a list of translations, called the *S0 page table.* The page table is a vector of translations containing one entry for each page, starting at page 0 of the region. This page table resides at a well-known address in *physical* memory, so the hardware can directly access the desired address to perform a translation. Figure 1 illustrates this process.

Process-space (P0) translations are only one step harder, since the P0 page table has the same structure as the S0 table, except that in order to save physical memory, it resides at a well-known address in the S0 region of the *virtual* memory space. Thus, to translate a P0 address, it is necessary to have the translation for the S0 page that contains the P0 page-table entry. This requirement implies that when a P0 TB miss is handled, the P0 PTE reference may cause another TB miss. This need for two page-table look-ups is referred to as a *double miss.* Note that further nested misses cannot occur, because the S0 page table resides at a *physical* memory address, which requires no translation. Figure 2 illustrates a P0 translation.

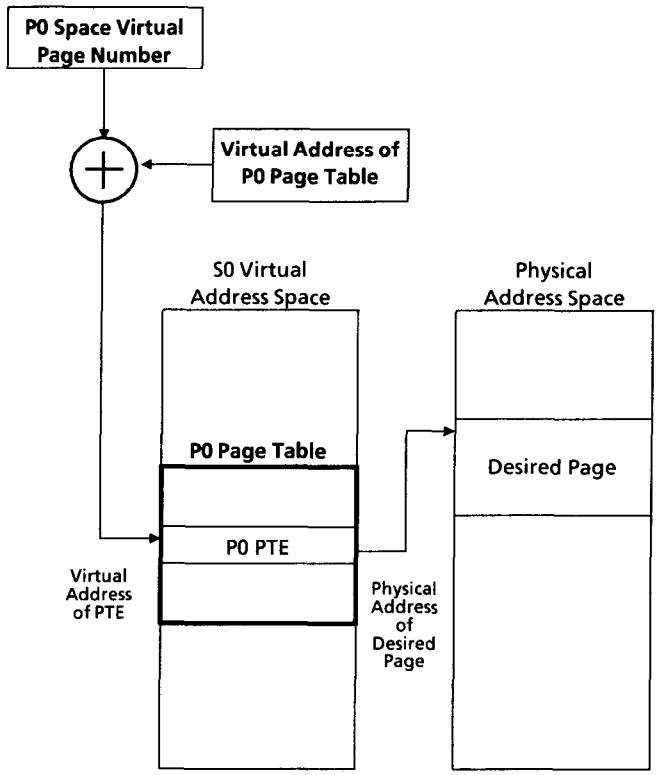

Fig. 2. Process space virtual page translation.

P1 address translation is nearly identical. In this case the well-known address points to a page table that is expected to grow backward (from high-numbered pages to low-numbered pages). This convention was adopted because the high end of the P1 region is expected to contain a large stack which grows downward.

One final characteristic of the VAX virtual memory architecture is that the S0 address space is common to all processes, while each process has distinct P0 and P1 address spaces. This has important ramifications for the implementation of the TB, because the processor must guarantee that the P0 and P1 translations of one process are not mistakenly used to translate addresses for another process. This is typically handled by removing all process-space translations from the TB on each process context switch; we will call this operation a *flush.*

The flush operation can have a significant impact on TB performance. Upon resuming execution of a process there will be "extra" misses that result solely from filling entries that had been flushed. Two other operations of the VAX result in the flushing of TB entries: TB Invalidate Single (TBIS) and TB Invalidate All (TBIA). These operations, which are invoked by special VAX instructions, result in either a single mapping or the entire TB being flushed. It is important to quantify the frequency of these various TB flushes to understand their impact on TB performance.

For a complete description of the VAX/VMS virtual memory structure, see [7, 13].

2.2 VAX-11/780 TB

The 11/780 TB is structured in the same manner as the tag-matching mechanism associated with main memory data caches [19]. Figure 3 illustrates this. The TB consists of a number of *rows*, where each row consists of a set of virtual-to-physical translations. To perform a translation, the virtual address is divided into two fields. The first field consists of the low-order bits of the address and is used to select the byte within the page. This number, called the *offset*, does not participate in the translation process but is retained to access the proper byte within the physical page selected. The other field consists of the virtual page number of the desired reference. It is used to determine if the translation is present in the TB. The page number is divided into two fields. The first, called the *row index*, is used to select a row within the TB. If the desired translation is to be found in the TB, it must be contained in this row. The other field is called the *tag* and contains sufficient information to identify uniquely a virtual address among all those that may reside in the selected row. This tag is compared against all the tags of the translations currently stored in the selected row of the TB. A translation hit occurs if there is a tag match. In this event the physical address of the page is extracted from the TB and combined with the saved offset to create the address for the desired reference to physical memory.

The structure of a TB is defined by the number of *rows* it contains and the number of tag entries that are compared in parallel when a row is selected, usually referred to as the *set size* or *associativity* of the TB. The total number of translations in the TB is referred to as the *size* of the TB.

Different implementations of the VAX architecture can have different types of translation buffers. The VAX model 11/780 TB considered in this paper has the following characteristics: its size is 128 translations; its structure is 2-way set-associative; and it uses random replacement on a miss. The 128 translations are evenly divided between system space and process space. Thus the 11/780 TB appears to be two separate translation buffers, each 64 entries in size, and each devoted to one address space. Other VAX implementations (all divided equally between system and process space) are as follows:

Model	Size (translations)	Associativity
11/780	128	2-way
11/750	512	2-way
11/730	128	1-way
11/785	512	2-way
8600	512	1-way

As we noted above, each entry of a VAX TB maps one 512-byte page. For any fixed size TB, bigger pages are always better: more memory is mapped at no added cost, resulting in fewer misses. Bigger is not always better, however, for overall paging behavior. The choice of page size is a trade-off involving the TB as well as other hardware and software components of the virtual memory system.

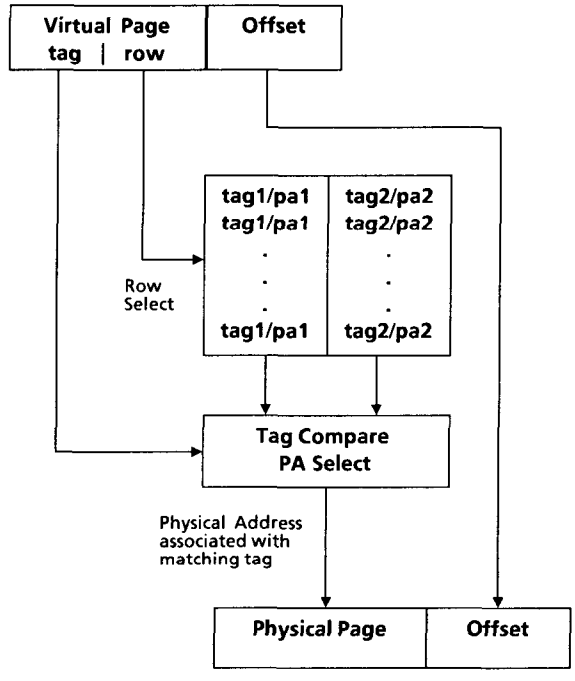

Fig. 3. 11/780 TB configuration.

Figure 4 is a simplified picture of the 11/780 address paths relevant to the TB. Translation requests are made both by the CPU for operand data (referred to as *D-stream* references) and by the 8-byte prefetching Instruction Buffer or IB (referred to as *I-stream* references). Every D-stream reference needs to use the TB, but since I-stream prefetching is strictly sequential, one I-stream reference to the TB can cover a set of consecutive references to memory. The Virtual Instruction Buffer Address register (VIBA) holds the prefetch pointer, and the Physical Instruction Buffer Address register (PIBA) holds its corresponding physical address. The IB's references to memory use the PIBA directly, bypassing the TB (see Figure 4). This works until one of two events interrupts sequential prefetching: a branch instruction or a sequential prefetch across a page boundary. A branch requires a reload of VIBA, and both branches and page crossings result in a TB reference to reload the PIBA.

The presence of the VIBA–PIBA registers complicates the task of measuring TB performance. The usual measure, namely, the *miss ratio*, is the number of misses divided by the number of references. In the 11/780's case, what exactly should be denoted by an I-stream "reference"? We discuss this question in the next section.

The technology of the 11/780 is 1975–76 vintage Schottky-TTL SSI and MSI. The processor is implemented on twenty 12-inch-by-15-inch boards (five more boards implement the Floating Point Accelerator). About one and one-third boards are devoted to the TB and its attendant hardware, with one-third of a board for the tag and data RAMs alone. Thus, about 6 or 7 percent of the hardware cost of the processor (neglecting memory and I/O) is spent for the TB.

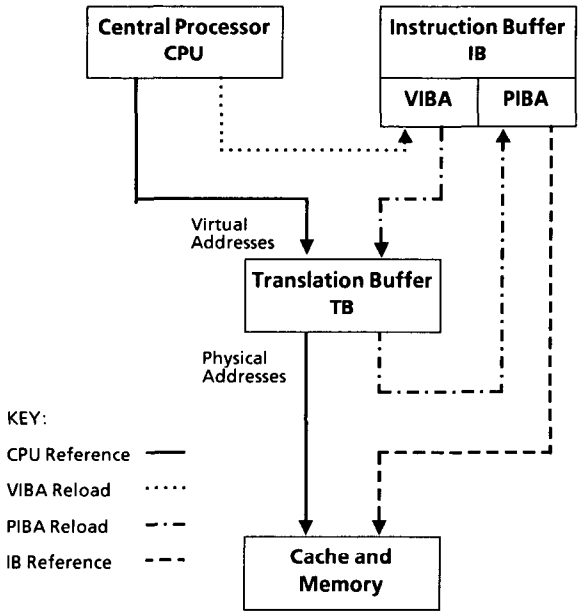

Fig. 4. VAX - 11 / 780 address paths.

Filling of TB entries on the 11/780 is performed by CPU microcode routines. Handling of D-stream reference TB misses is simplest, since these routines are entered via a microcode trap when the miss occurs. I-stream misses are more complex. The IB hardware first sets a state bit indicating that it received notification of an I-stream TB miss. Eventually the CPU finds an insufficient number of bytes in the IB, notices that the bit is set, and calls the microcode routine to do the fill.

2.3 Performance Metrics

We will use two different performance metrics in this paper. The first is the simple miss ratio. For the D-stream, this is just the number of misses divided by the number of data references made by the program. In the 11/780, data references are all for a 4-byte *longword* of data, and therefore data items that are longer than a longword will result in multiple references, as will references to any data item that crosses a longword boundary.

For the I-stream miss ratio, we will follow the hardware. The TB miss ratio will be the number of misses caused by the IB divided by the number of TB requests (reloadings of PIBA) made by the IB. This makes these I-stream numbers 11/780-specific; other implementations of the VAX architecture might have quite different I-stream TB miss ratios, even with a similar TB. The difficulty in determining the exact number of references makes comparison of miss ratios especially difficult, and the difficulty can be compounded when trying to contrast performance between machines with different architectures.

A second metric will help eliminate some of this dependence on the specific processor organization. This metric is the number of TB misses *per instruction.* Although this metric is clearly VAX-specific, since VAX instructions are different

from the instructions of other architectures, it does allow easy comparison with other VAX implementations by abstracting some of the organization-specific characteristics that are not related to TB performance. For example, comparison of widely varying instruction buffer structures will show nearly identical numbers of misses per instruction, even if one structure repeatedly reads the same location while the other does not. Similarly, the misses per instruction metric will not be as sensitive to data-path access widths as misses per hardware reference. Finally, we have found that this metric, along with the average number of cycles per instruction and the number of cycles to service a TB miss, allows simple computation of the relative importance of the TB miss service time to processor performance. (See [1, 12] for other uses of this metric.)

In order to make both of our metrics easier to deal with, we will multiply them by 100, reporting misses per 100 instructions and miss ratios as percentages between 0 and 100.

3. METHODS

Apart from analytic modeling, there are two approaches to understanding the performance of a computer system component such as a TB: simulation and measurement. Both approaches have advantages and disadvantages.

Simulation requires no special hardware and is quite flexible, since variations in hardware structure can easily be evaluated. Simulation experiments are reproducible. On the other hand, simulation can be inaccurate if the structure of the simulated model does not closely match the structure of the actual hardware. Certain aspects of performance may require an impractical amount of detail in the simulation. It may also be difficult to create an accurate representation of the real work load, especially if that workload must include the interactions of many distinct instruction streams. Simulation is also expensive in computational resources: simulated hardware typically runs orders of magnitude slower than real hardware and requires large amounts of storage.

Direct measurement of the hardware has complementary advantages and disadvantages. There is no question concerning accuracy, of course, nor is there any problem with the speed of the experiments. But variations in the hardware structure—an enlargement of the TB, for example—are generally not possible to measure, and measurements of a real work load are not reproducible. Perhaps the biggest difficulty with measurement is the inescapable requirement for some sort of instrument and the need to connect it correctly but harmlessly to the system under test.

Three independent TB studies done at Digital attempted to characterize the 11/780 TB in support of new VAX processor development. Both measurements and simulation techniques were used. The first study used a commercial hardware monitor, the second used a monitor specially designed and built for VAX measurements, and the third used trace-driven simulation. In the remainder of this section we will discuss these three experimental approaches in detail. The three techniques do not yield fully comparable sets of results, so our presentation of data may sometimes use results from one or two of them; where possible, however, we report results from all three experimental settings.

Common to all our experiments was the use of the VMS operating system (version 2) [6, 13] on 11/780s with Floating Point Accelerators. The VMS null

process, which runs when the system is idle, was excluded from all measurements. The null process was excluded because its trivial code structure (branch to self, waiting for an interrupt) results in no TB misses and would therefore bias the results in proportion to the amount of time the CPU was idle.

3.1 Plugboard Hardware Monitor

A Dynaprobe™ Model 7916 hardware monitor (manufactured by NCR COMTEN, Inc.) was used in the first set of experiments. Seventeen measurement probes were attached to 11/780 backpanel pins and in three instances to new signals created with added logic on one processor board. The Dynaprobe has a logic plugboard that was used to refine these signals into the quantities actually counted.

The Dynaprobe was used in two different experimental settings. In the first, the Dynaprobe was simply run all day long, measuring whatever happened to be running on the VAX that day. The particular VAX we measured was an 11/780 used at a Digital engineering site. This machine was used for general-purpose time-sharing, including editing, program development, simulation, electronic mail, and so on. Measurements were taken on three days; the three sets of results will be called Dyn-DAY1, Dyn-DAY2, and Dyn-DAY3 (DAY for DAY-long measurement). During the period of these experiments, roughly 30 users were logged on in the middle of the day.

The only major parameter that varied across the experiments was the main memory size. For the Dynaprobe time-sharing measurements, an 11/780 with 5 Mbytes of interleaved memory was used.

Although realistic, these experiments are not repeatable, since the computational load varies greatly over time. A second experimental setting addressed this problem. In it, a Remote Terminal Emulator (RTE) [9, 21] provided a real-time simulation of 40 time-sharing users connected to the VAX. The RTE is a PDP-11 with 40 asynchronous terminal interfaces; output characters generated by the RTE from canned user scripts are seen as terminal input characters by the VAX, and vice versa. The work load was designed to simulate a general time-sharing load in an educational program-development environment. Some of the simulated users were editing, compiling, linking, and executing programs in various languages; some were updating indexed files; and some were doing numerical computation. For these experiments the 11/780's memory was reduced to 3 Mbytes. In later experiments with another hardware monitor we used the RTE to emulate an engineering and commercial environment; these are described later. Five-minute segments of this work load were measured. The two repetitions of this experiment will be called Dyn-EDU1 and Dyn-EDU2.

In the Dynaprobe measurements PDP-11 compatibility mode was excluded along with the VMS null process. Compatibility mode was excluded because some of the hardware signals measured do not accurately reflect compatibility mode execution.

3.2 Micro-PC Histogram Measurement

A second set of measurements were collected via a special-purpose hardware monitor that enabled us to create histograms of microcode execution in the VAX-11/780 processor. This micro-Program Counter, or μPC, monitor consists of a general purpose histogram count board that has 16,000 addressable count loca-

tions (or histogram buckets) and is capable of incrementing the count in a selected location at a rate commensurate with the microcode execution rate of the VAX-11/780. In conjunction with the histogram count board, a processor-specific interface board was required to provide the address of a histogram count bucket and control lines to signal when a count should be made. For these experiments the interface board addressed a distinct histogram bucket for each microcode location in the processor's control store, and a count was taken for each microinstruction executed. The capacity of the counters was sufficient to collect data for 1 to 2 hours of heavy processing on the CPU.

The histogram collection board was designed as a UNIBUS device, and UNIBUS commands were used to start and stop data collection, as well as to clear and read the histogram count buckets. Coincidentally, since the 11/780 has a UNIBUS, the histogram collection monitor could be installed directly on the system being measured, obviating the cost and nuisance of using a second machine for the hardware monitor. This was a further convenience, as the data collected was immediately available on a machine of sufficient capacity to do the data reduction. Note, however, that while actually monitoring microcode execution, the data collection hardware is totally passive, causing no UNIBUS activity and having no effect on the execution of programs on the system.

Since much of the activity in the 11/780 processor is under direct microcode control, the frequency of many events can be determined through examination of the relative execution counts of various microinstructions. Of particular interest for this study were the characteristics and service times of various activities associated with translation buffer fills and flushes.

The μPC histogram data is especially useful, since it forms a general resource from which the answers to many questions concerning the operation of the 11/780 running the same work load can be obtained simply by doing additional interpretation of the raw histogram data [8].

The major disadvantage of this method of hardware monitoring is that certain hardware events are not visible to the microcode. For example, the counts of instruction-stream memory references are not available because they are made by a distinct portion of the processor not under direct control of the microcode. Another characteristic of this measurement technique is that the analysis produces only average behavior characterizations of the processor over the measurement interval, since no measures of the variation of the statistics during the measurement are collected.

The μPC histogram measurements were taken in two different experimental settings: live time-sharing and synthetic RTE-generated loads. The live time-sharing measurements were taken from two different machines within Digital engineering. The first machine belonged to the research group and was used for general time-sharing and some performance data analysis. The general-purpose time-sharing was similar to that measured with the Dynaprobe, consisting of text-editing, program development, and electronic mail. This machine was relatively lightly loaded and had approximately 15 users logged in during the measurement interval. Measurements taken on this machine will be referred to as μPC-TS1 (TS for time-sharing).

The second time-sharing measurements were taken from a machine being used by a group in the initial stages of development of a VAX CPU. The load on this

machine consisted of the same type of general purpose time-sharing as in μPC-TS1, with the addition of some circuit simulation and microcode development. This machine had a heavier load, with approximately 30 users logged in during the measurement interval. Measurements taken on this machine will be referred to as μPC-TS2.

The remaining measurements were taken under RTE-generated loads. These loads provided a more repeatable measurement environment and were used to test the system in environments different from those available inside Digital engineering. The three measurements of this type were taken in an educational environment, an engineering/scientific environment, and a commercial transaction-processing environment, respectively. The educational environment, which we will refer to as μPC-EDU, is the same load as used for the Dynaprobe measurements. The engineering load consisted of 40 users doing program development and scientific computations. This load will be called μPC-SCI. And finally, the commercial load consisted of 32 users doing transactional database inquiries and updates. This load will be denoted μPC-COM.

As with the Dynaprobe measurements, the VMS null process was excluded. The PDP-11 compatibility mode, however, was not. The percentage of time each measurement spent in PDP-11 mode was as follows:

TS1:	9.5 percent
TS2:	3.3 percent
EDU:	0.3 percent
SCI:	4.0 percent
COM:	0.3 percent

All of these experiments ran on machines with 8 Megabytes of memory, and each experiment lasted about one hour. Note that the Dyn-EDU1 and EDU2 experiments differ from μPC-EDU in memory size and measurement duration.

3.3 Trace-Driven Simulation

Trace-driven simulation works as follows. A program of interest is executed interpretively, and a record is made of each of its memory references. The *address trace* that results is then used to drive a simulation of the translation buffer under study. Instruction tracing features are provided in the VAX architecture, so it was relatively easy to generate address traces [22].

The major disadvantages of simulation fall into three major categories:

(1) limitations in the scope of the address trace itself;
(2) the inability to properly model multiprocess effects; and
(3) the cost of modeling the performance of a sufficient number of program traces to achieve a "representative" measure of the performance of the translation buffer, especially in those cases where detailed modeling of the hardware is required to assess the interactions and impact of TB references from various hardware units.

Our simulation effort strove to overcome some of these difficulties. We will consider each in turn.

Most trace-driven simulations have been based on traces obtained from the user-mode execution of the subject program. Since it is easiest to do simulations directly from a user program image or to do trace traps during the user-mode execution of a program, these techniques have been most popular for the generation of address traces. Unfortunately, except in cases of batch-style computation-intensive applications, much of the processor's time is spent within the operating system. This time is spent doing I/O processing, system services, and interrupts. Therefore, we developed a trace technique that allows tracing to occur during all modes of processor operation and that therefore enabled us to capture much more of the processor's activity. This trace facility allowed us to capture all the system service execution with the exception of certain time-dependent interrupt service operations, and instruction faults. Of course, slowing down the subject program while tracing it was unavoidable, and consequently the relative timing between the program and its I/O was changed. This in turn could change the context-switching behavior of the program.

The second difficulty with trace generation is the inability to incorporate accurately the effect that other processes have on the execution of the process being simulated. The primary manifestation of this effect is the fact that the process being simulated may be removed from execution temporarily, and the state that it has built up in the TB will be lost. Interrupts also result in short pauses in program execution.

Pauses in program execution can be defined as falling into two categories: voluntary and involuntary waits. Voluntary waits are those pauses in execution that are a direct consequence of program execution. Examples of voluntary waits are waiting for disk or terminal I/O or page-fault service. The typical technique of adding TB flushes at random intervals will not properly model these program-dependent pauses. However, since our multimode instruction traces include all the code leading up to most I/O waits, including the VAX save process context instruction (SVPCTX), our TB simulator can detect and model these pauses properly. Page-fault activity is more difficult to obtain, since it can be load dependent, and we felt that small traces would have an inordinate number of page faults for their size. Therefore in our simulations we did not explicitly calculate where page faults would occur, but treated them as a component of the involuntary waits.

Involuntary waits result primarily when a higher priority process preempts the CPU. This event is basically load-dependent, and therefore its frequency was set as a parameter in the simulation with a value determined from other measurements. In addition, a process that has exceeded the limit of its time slice on the CPU is removed from execution. The frequency of this event is a system parameter that is easily incorporated into a simulation. In practice on VMS systems, the process time-out is relatively infrequent and was not considered.

In general, each time a program is removed from execution there is the opportunity for the state of the TB to be lost to some degree. The loss may be almost total if there is a process context switch, which causes the process-space section of the TB to be flushed, and sufficient references are made to S0 space that its contents are completely changed. On the other hand, the loss may be minimal if the cause was a short burst of interrupt processing. In the simulations

we ran, it was assumed that any context switch resulted in a complete flush of the process half of the TB but had no effect on the system half.

The traces we used in these studies were derived from a number of Digital utilities available under VMS. A study conducted by Jain [11], which characterized the work loads at a number of VAX sites at educational institutions, has shown that on average more than half of all computation time was spent in system programs. Therefore, considering the limited number of benchmarks that could be run, we selected some of the more heavily used system utilities in an effort to obtain a set of traces that would be representative of a large fraction of the work on such VAX/VMS systems.

The utility traces that we used were:

DIR A directory listing of the files in a user's file directory (47,212 instructions)
FORT A FORTRAN compilation (2,293,007 instructions)
LINK The LINKing of a program (481,301 instructions)
MAIL The interactive sending and receiving of electronic mail messages (284,986 instructions)
EDT The interactive use of the VAX/VMS standard screen editor (1,050,746 instructions)
RNO The execution of a text processing utility (2,704,570 instructions)
SORT The SORTing of an ASCII file (2,158,468 instructions)

The simulation experiments will be referred to as Sim-DIR, Sim-FORT, etc.

4. TB MISSES

This section will present most of the basic TB miss data. We look first at miss ratios, the most common metric of TB performance, and then proceed to consider more detailed data and other performance metrics.

4.1 Miss Ratios

Table I presents miss ratios for the Dynaprobe experiments, calculated separately for I-stream versus D-stream and for process space versus system space. (As we mentioned above, I-stream miss ratios are only available in the Dynaprobe data.) Row, column, and overall totals represent miss data pooled over the corresponding categories; for example, the total I-stream miss ratio is calculated without distinguishing process from system activity. Recall that the 11/780 TB is split into process and system halves, each of which holds a mixture of I-stream and D-stream translations.

These miss ratios are strikingly and consistently different across the four categories. The I-stream miss ratios are all greater than the corresponding D-stream miss ratios, particularly in system space. Part of the explanation for this lies in our definition of the I-stream miss ratio: only branches and page-crossings are counted as TB references.

System space miss ratios are much higher than process space miss ratios, particularly in the I-stream. We offer three informal hypotheses:

(1) System code and data structures are simply bigger than process code and data structures. In an average locality interval, system code involves more

Table I. Miss Ratios in the Dynaprobe Experiments (Percent)

		I-stream	D-stream	Total
Process	Dyn-DAY1	1.0	0.8	0.9
	Dyn-DAY2	1.5	0.9	1.1
	Dyn-DAY3	0.7	0.6	0.7
	Dyn-EDU1	1.1	1.0	1.0
	Dyn-EDU2	1.1	0.9	1.0
System	Dyn-DAY1	19.4	4.8	6.9
	Dyn-DAY2	17.5	4.0	5.8
	Dyn-DAY3	15.4	5.4	7.2
	Dyn-EDU1	31.7	6.2	9.5
	Dyn-EDU2	32.5	6.7	10.0
Total	Dyn-DAY1	3.9	1.7	2.2
	Dyn-DAY2	2.9	1.4	1.8
	Dyn-DAY3	3.5	1.6	1.9
	Dyn-EDU1	6.5	2.4	3.2
	Dyn-EDU2	5.9	2.4	3.1

pages than process code and will have higher TB miss ratios in both I-stream and D-stream.

(2) In addition to being larger, system data structures are more complex than process data structures. They are more likely to be pointer-rich structures such as queues and linked lists, and therefore are more likely to involve multiple pages. A higher D-stream miss ratio results.

(3) System loops have fewer iterations than process loops. I-stream processing only needs the TB for branches and page-crossings, so once a loop's pages are in the TB, further iterations are highly unlikely to cause TB misses. The greater the number of iterations, the lower the I-stream miss ratio for that part of the program.

Although some of these miss ratios look quite large, the overall percentages (bottom right corner of the table) are reasonably small: two to three percent. This, of course, is due to the distribution of TB *references* across the four categories. Figure 5 shows the distribution of references and misses across the four categories for Dyn-DAY3; the other experiments are similar. Process references account for more than three-quarters of all references, but because of their lower miss ratios, they account for less than one-third of the misses.

4.2 Misses per Instruction

In this section we examine the TB miss behavior using the misses-per-instruction metric. Tables II–IV tabulate misses per instruction for the Dynaprobe, μPC monitor, and simulation experiments, respectively. Just as above, misses are categorized as arising from I-stream or D-stream references, and from process-space or system-space references. The μPC measurements are an exception, since the process-space/system-space distinction was not available.

Note that these miss rates are not relative to the number of references of each type, or to the type of instruction issuing the reference, but are simply the number of misses over all instructions.

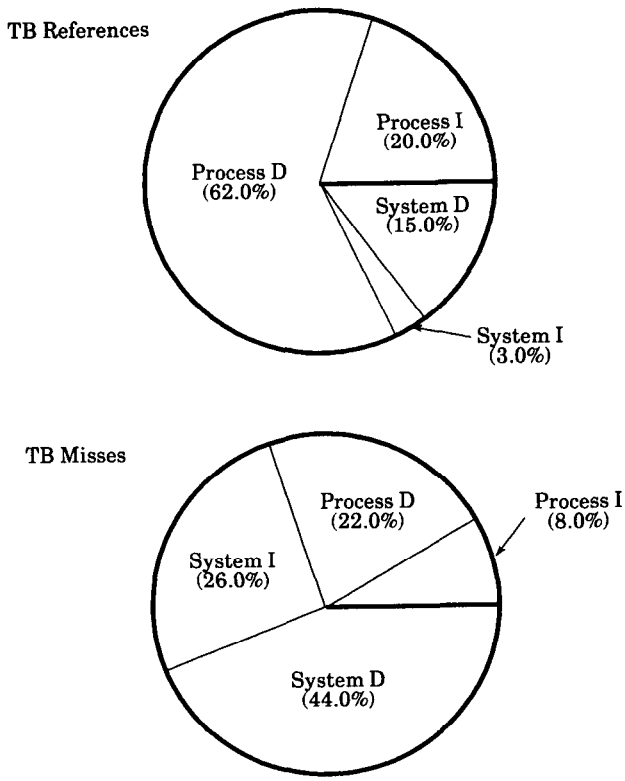

Fig. 5. Distribution of references and misses in Dyn-Day3.

Table II. TB Misses per 100 Instructions (Dynaprobe)

		I-stream	D-stream	Total
Process	Dyn-DAY1	0.23	0.73	0.96
	Dyn-DAY2	0.42	0.73	1.15
	Dyn-DAY3	0.18	0.50	0.69
	Dyn-EDU1	0.23	0.77	1.00
	Dyn-EDU2	0.24	0.77	1.00
System	Dyn-DAY1	0.87	1.26	2.13
	Dyn-DAY2	0.48	0.69	1.17
	Dyn-DAY3	0.62	1.03	1.64
	Dyn-EDU1	1.44	1.91	3.34
	Dyn-EDU2	1.32	1.88	3.21
Total	Dyn-DAY1	1.10	1.99	3.10
	Dyn-DAY2	0.90	1.41	2.32
	Dyn-DAY3	0.80	1.53	2.33
	Dyn-EDU1	1.67	2.67	4.34
	Dyn-EDU2	1.56	2.65	4.21

Table III. TB Misses per 100 Instructions (μPC)

		I-stream	D-stream	Total
Total	μPC-TS1	0.92	2.10	3.02
	μPC-TS2	0.83	2.12	2.95
	μPC-EDU	0.99	2.12	3.11
	μPC-SCI	0.70	1.80	2.50
	μPC-COM	1.24	2.18	3.41

Table IV. TB Misses per 100 Instructions (Simulator)

		I-stream	D-stream	Total
Process	Sim-DIR	0.17	0.57	0.73
	Sim-FORT	0.40	1.42	1.82
	Sim-LINK	0.46	1.41	1.87
	Sim-MAIL	0.15	0.54	0.69
	Sim-EDT	0.27	1.05	1.32
	Sim-RNO	0.50	1.37	1.87
	Sim-SORT	0.12	0.51	0.63
System	Sim-DIR	1.04	0.92	1.96
	Sim-FORT	0.02	0.02	0.04
	Sim-LINK	0.54	0.94	1.48
	Sim-MAIL	1.39	1.37	2.75
	Sim-EDT	0.70	1.02	1.72
	Sim-RNO	0.03	0.05	0.07
	Sim-SORT	0.15	0.38	0.53
Total	Sim-DIR	1.21	1.48	2.69
	Sim-FORT	0.41	1.45	1.86
	Sim-LINK	1.00	2.35	3.35
	Sim-MAIL	1.54	1.91	3.44
	Sim-EDT	0.97	2.07	3.04
	Sim-RNO	0.53	1.41	1.94
	Sim-SORT	0.27	0.89	1.17

The misses-per-instruction metric gives a good indication of the relative impact of TB misses on the processor performance, since the number of misses determines the amount of time expended in filling TB entries. In fact, the hardware monitor measurements show remarkable consistency in the relative impact of each reference type. The D-stream miss rate was consistently greater than the I-stream miss rate for both process space and system space. In 100 instructions in a time-sharing load, there are roughly two D-stream misses and one I-stream miss. This implies that there is greater locality in the I-stream than in the D-stream. Therefore, if one were to consider distinct TBs for I-stream and D-stream references, a smaller one could be used for the I-stream.

The system-space miss rate was consistently greater than the process-space miss rate regardless of whether the reference was from the I-stream or D-stream. This is particularly interesting, since the system portion of the TB is not flushed

Table V. Types of TB Misses (Percent)[a]

	P0	P1	Double	S0
μPC-TS1	46.7	8.2	3.1	42.0
μPC-TS2	31.0	6.2	3.4	59.4
μPC-EDU	19.4	9.0	3.4	68.1
μPC-SCI	28.5	5.0	3.0	63.4
μPC-COM	16.4	9.2	4.8	69.6

[a] This table actually shows the types of *PTE references*, which are very nearly the same as the types of TB misses. See the text for details.

on context switches. As we hypothesized above, this may be a consequence of the types of data structures and program structures typical of system code. But it also implies that it may be most cost-effective to increase the size of the system portion of the TB so that it is larger than the process portion.

4.3 Double Misses

As described in Section 2.1, the virtual memory of the VAX is divided into three regions, the system space, S0, and two regions of process space, P0 and P1. Furthermore, the page tables for the process-space regions are held in the S0 region of the virtual address space to avoid allocating physical memory for the page tables of every process. However, this results in the possibility of *double* TB misses. To assess the impact of these misses, we used the μPC histograms to measure the relative frequency of the page table look-ups from each of the following reference sources: P0 space, P1 space, S0 space due to double misses, and normal S0 space references. (Practically all of these look-ups were for TB misses; the rest were for protection-checking probe operations used in some VAX instructions.) Table V summarizes these results.

The table shows again the large proportion of TB misses attributable to system-space references. It also shows that P0 misses in turn greatly outnumber P1 misses. This we presume is due to the fact that P1 holds the stacks, which have inherently good locality of reference. Finally, the table shows that only a small number of page table references were for double misses. This implies that the allocation of process page tables in virtual memory had only a small cost in terms of multiple page-table translations.

4.4 I-stream Details

I-stream TB misses have two sources: explicit branches in the program and prefetches that cross a page boundary. There are two slightly incompatible observations of this distinction to report. A miss due to page-crossing can happen on any byte of an instruction. A VAX instruction consists of an opcode byte followed by a variable number of operand specifiers of various sizes [7, 20]. There are three types of page-crosses: on the opcode byte itself, on any byte of the first operand specifier, and on any byte of any later specifier. The μPC method could not distinguish branch-type misses from those due to the first two types of page cross; results are shown in Table VI. The simulation runs, on the other hand, could clearly distinguish between branch misses and all page-cross misses; these results are shown in Table VII.

Table VI. Type of I-stream Misses in μPC Experiments (Percent)

	Branch target	Page-crossing	
	Opcode	Opcode	First specifier	Other specifier
μPC-TS1		93	7
μPC-TS2		94	6
μPC-EDU		95	5
μPC-SCI		94	6
μPC-COM		95	5

Table VII. Type of I-Stream Misses in Simulation Experiments (Percent)

	Branch target	Page-crossing	
	Opcode	Opcode	First specifier	Other specifier
Sim-DIR	91	9
Sim-FORT	76	24
Sim-LINK	64	36
Sim-MAIL	86	14
Sim-EDT	82	18
Sim-RNO	68	32
Sim-SORT	74	26

Not surprisingly, most I-stream misses are due to branches, which therefore become a more important candidate for efficient service of TB misses than I-stream page crossings.

4.5 D-stream Details

The μPC and simulation experiments allowed us to separate D-stream TB references and misses into reads and writes. Table VIII shows the miss ratios of read and write references from the D-stream. Modify references here count as one read followed by one write, where the write will never miss. Dynaprobe D-stream miss ratios were presented in Table I, and they are similar to the ones given here.

Reads appear to be two to four times as likely to miss as writes. A simple hypothesis to explain this is that within a locality interval, data is more likely to be written after it is read than before. Thus the reference that brings an entry into the TB is more likely to be a read than a write.

4.6 Modify Bit Characterization

In order to support the paging strategy of the operating system, the VAX architecture provides help in indicating which pages have been modified. A PTE bit called the *M-bit* (modify bit) is set whenever a page that has not yet been marked as modified is written into. There is an M-bit associated with each virtual page, and it resides in the page-table entry for the page. A copy of the M-bit resides in the TB entry for any page whose translation is present in the TB.

Table VIII. D-Stream Miss Ratios (Percent)

	Read	Write	Total
μPC-TS1	2.13	1.11	1.76
μPC-TS2	2.58	0.64	1.93
μPC-EDU	2.17	0.79	1.67
μPC-SCI	1.75	0.78	1.43
μPC-COM	2.34	0.76	1.80
Sim-DIR	2.28	0.71	1.65
Sim-FORT	1.73	0.54	1.25
Sim-LINK	2.55	1.17	1.99
Sim-MAIL	2.67	0.73	1.89
Sim-EDT	2.32	0.70	1.67
Sim-RNO	1.46	0.61	1.13
Sim-SORT	0.93	0.60	0.83

Table IX. M-Bit Setting Rate (Settings per 100 Instructions)

	Write with TB miss	Write with TB hit	Total
μPC-TS1	0.0095	0.0022	0.0117
μPC-TS2	0.0011	0.0007	0.0018
μPC-EDU	0.0031	0.0016	0.0047
μPC-SCI	0.0042	0.0026	0.0067
μPC-COM	0.0026	0.0017	0.0044

Setting the M-bit can occur in two circumstances, depending upon whether the current reference has resulted in a TB miss or a TB hit. The latter case arises if at some time in the past the page was read and a translation put into the TB, but the current reference is the first to modify the page. Setting the M-bit in cases of TB misses is easy, since a check can be incorporated in the TB fill service. Setting the M-bit on TB hits is handled as a special microcode exception case on memory references.

M-bit setting operations are overheads incurred for virtual memory support, and their performance impact is proportional to their frequency of occurrence. We used the μPC histogram measurements to find the frequency of setting the M-bit. Table IX shows the average rate of M-bit settings, for cases of both TB hits and misses.

It is clear from the table that the frequency of M-bit settings is two orders of magnitude less than the frequency of TB misses, and consequently the few micro-operations required to set the M-bit will have an insignificant effect on 11/780 performance.

5. TIME IN TB MISS SERVICE

In the VAX-11/780, TB misses are handled by microcode routines that read the required PTE from memory and insert it in the TB. The time spent doing this is a major performance cost of the TB: cycles so spent are lost to the running program. (If the PTE indicates that the page is not currently in main memory,

Table X. Time in TB Miss Service
(Average Cycles per Miss)

	Nonstalled	Stalled	Total
Dyn-DAY1	—	—	21.5
Dyn-DAY2	—	—	21.7
Dyn-DAY3	—	—	21.8
Dyn-EDU1	—	—	22.1
Dyn-EDU2	—	—	22.1
μPC-TS1	18.6	2.4	21.0
μPC-TS2	18.1	3.5	21.6
μPC-EDU	18.0	3.8	21.8
μPC-SCI	18.2	3.8	21.9
μPC-COM	17.6	4.0	21.7

i.e., a *page fault*, then the TB miss microcode must yield control to memory-management software, which will bring it in. Time spent this way is a major performance cost of paged virtual memory and is a function of the size of main memory, speed of the disk, memory-management algorithm, and so on. See [14] for a discussion of VAX/VMS memory management.)

We measured TB miss time in the Dynaprobe and μPC experiments. The μPC histogram gives the result directly, reporting execution counts for the microinstructions in the TB service routines. In the Dynaprobe experiments we marked the entries and exits of these routines by using hardware intended to mark the locations of microcode patches. The Dynaprobe counted cycles between entry and exit. Cycles in the 11/780 are 200 nanoseconds long.

The TB miss routine must locate the appropriate page table, read the proper PTE from memory, and write the mapping into the TB. The routine in its simplest form is roughly 17 microinstructions long; there is some minor additional overhead (one or two cycles) on entry and exit. The PTE, of course, may not be in the *cache*. Hence reading it may provoke a cache miss and a processor wait, or *stall*, of about 6 cycles, depending on other simultaneous memory activity [3]. The μPC measurements separated the TB routine's cycles into stalled and nonstalled cycles; the Dynaprobe measurements did not.

Table X shows the results. These results include the routine entry and exit overhead, and double misses count here (as throughout this paper) as two TB misses. According to the table, TB misses take about 22 cycles each, including the cache stall and the overhead. The average length of the cache stall is 3.5 cycles. Since a cache miss usually takes 6 cycles, we conclude that the PTE is, on average, in the cache about 40 percent of the time.

What is the performance cost of the TB service time? Combining the time-per-miss results and miss-rate measurements of Section 4.2 with a knowledge of the total time required for instruction execution provides an understanding of the amount of time attributable to TB service and its relation to the total execution time of an instruction. Since TB service time is added explicitly to time for base instruction execution, the effect of TB service on the 11/780 can be represented by the simple formula,

$$\text{cpi}_{\text{total}} = \text{cpi}_{\text{base}} + \text{cpi}_{\text{tb}},$$

Table XI. Time in TB Miss Service

	VAX MIPS	cpi_{base} +	cpi_{tb} =	cpi_{total}	Service time as percent of all cycles
Dyn-DAY1	0.46	10.08	0.67	10.75	6.2
Dyn-DAY2	0.50	9.42	0.50	9.92	5.1
Dyn-DAY3	0.51	9.31	0.51	9.82	5.2
Dyn-EDU1	0.42	11.00	0.96	11.96	8.0
Dyn-EDU2	0.43	10.81	0.93	11.74	7.9
μPC-TS1	0.52	8.97	0.63	9.60	6.6
μPC-TS2	0.50	9.28	0.64	9.92	6.4
μPC-EDU	0.45	10.53	0.68	11.21	6.0
μPC-SCI	0.50	9.51	0.55	10.06	5.5
μPC-COM	0.46	10.15	0.74	10.89	6.8

where

$$cpi_{tb} = m_{tb} \times t_{service}$$

and

cpi_{total} = total cycles per instruction,
cpi_{base} = cycles per instruction exclusive of TB miss service,
cpi_{tb} = cycles per instruction for TB service,
m_{tb} = TB misses per instruction (see Table II),
$t_{service}$ = TB miss service time in cycles (see Table X).

Table XI shows the execution rate of the 11/780 and its decomposition into the time spent in base execution and in TB service as determined in the Dynaprobe and μPC measurements. The first section of the table shows the execution rate measured in millions of VAX instructions per second (VAX MIPS). Note that this measure is specific to the VAX architecture and cannot be used to compare different architectures. The next section of the table shows the application of the above formula to determine the number of instruction cycles dedicated to TB service relative to total instruction execution. (A more detailed timing model of the 11/780 is given in [8].)

Enhancing the 11/780 TB structure does not, it appears, offer much leverage for increasing the performance of the processor. Even if there were no TB misses at all, performance could be improved by only 5–8 percent. And even a TB of infinite size could not realize this improvement because of the misses due to context-switching. In the next section we discuss these misses, and in Section 7 we look at some finite variations of the 11/780 TB structure.

6. CONTROL FLOW CHANGES

6.1 Context Switches

In the VAX architecture, a process context switch is accomplished by a save-process-context instruction (SVPCTX) followed by a load-process-context instruction (LDPCTX). In VMS, other methods with less overhead are used to

Table XII. Context Switches (LDPCTX) from Dynaprobe and μPC

	Average headway		Average number of process space misses per interval
	Time (ms)	Instructions	
Dyn-DAY1	11	5,140	82
Dyn-DAY2	17	8,600	152
Dyn-DAY3	14	7,350	64
Dyn-EDU1	7.8	3,250	35
Dyn-EDU2	8.0	3,400	37
μPC-TS1	13.7	7,100	83
μPC-TS2	11.2	5,660	83
μPC-EDU	12.7	5,660	49
μPC-SCI	21.3	10,600	89
μPC-COM	9.5	4,350	35

transfer control to and from the system and to service interrupts; full context switches happen only when one user process is replaced by another. A context switch changes process state, including the process virtual address space. This is typically accomplished by invalidating all of the process-space entries in the TB. Since context-switching is thus responsible for some portion of the TB misses, it is important to know how often it occurs. (The TB flush itself takes 40 cycles on the 11/780.)

Table XII describes context-switching in the Dynaprobe and μPC experiments. The average *headway* is the average length of the interval between switches, here measured in milliseconds and in VAX instructions (but, as usual, not counting the null process). The length of a context-switch interval is a complex function of system load and hardware configuration. These lengths vary widely. According to the table, there were from 3200 to 10,600 instructions in the average interval. The longest and shortest intervals were observed in the RTE experiments. In the real time-sharing runs the average interval ranged between 5100 and 8600 instructions.

Roughly speaking, the shorter the average context interval, the greater the number of misses per average instruction. This can be seen by comparing Table XII with Tables II and III. In particular, the worst TB performance is found in the Dyn-EDU workloads, which have the shortest average intervals. This we attribute to the fact that these experiments used the smallest amount of main memory.

Table XII also shows the number of process-space misses in the average interval. This turned out to be, to us, surprisingly low. Indeed, only one of the numbers shown in the table is much greater than the maximum number of process-space entries in the TB, namely 64. This led us to investigate, through simulation, how many misses could be attributed purely to filling up empty slots in the TB. (Unfortunately, neither hardware monitor allowed this to be measured directly.)

When the new process beings execution after a context switch, it will at first experience TB misses solely on account of the process TB being empty; these misses fill an empty slot and will be called *fills*. Once some fills have occurred,

Table XIII. Context Switches in the Simulations

	Average headway (instructions)	Flushes due to SVPCTX (percent)	Process-space misses per context interval		
			Fills	Bumps	Total
Sim-DIR	4,721	50.0	22.6	12.0	34.6
Sim-FORT	9,926	0.4	49.7	131.0	180.7
Sim-LINK	7,763	21.0	39.5	105.5	145.0
Sim-MAIL	6,064	38.3	29.1	12.9	42.0
Sim-EDT	1,812	81.7	16.4	7.5	23.9
Sim-RNO	9,729	2.5	47.8	133.9	181.7
Sim-SORT	9,679	3.1	25.2	36.1	61.3

the process can miss on an address that maps to an occupied TB slot; these misses replace one valid entry with another and will be called *bumps*. Every process-space miss is either a fill or a bump. We distinguish between these because only bumps can be reduced by enlarging or restructuring the TB. The number of fills in a particular context interval cannot be reduced, and may likely be increased, with a bigger or more associative TB.

As mentioned in Section 3.3, not all context switches could be recorded in our simulation traces. In particular, those due to VAX traps and faults (such as page faults) were not seen. To compensate for this, the simulated TB was arbitrarily flushed every 10,000 instructions, in addition to the flushes due to SVPCTX instructions in the traces.

Table XIII gives the results on context-switching in the simulations. There is fairly wide variation in all of the reported statistics. In particular, the total number of process misses per context interval has a wider range than the values seen in the measurements (Table XII). The three runs with the smallest percentage of flushes from SVPCTX are the three with the smallest amount of system activity (see Table IV).

The shorter the interval, the greater the ratio of fills to bumps is likely to be. Table XIII shows that the three programs with the shortest average intervals (DIR, MAIL, and EDT) are also the three in which fills actually outnumber bumps. In these three and in SORT, less than half of the available 64 entries were filled in the average interval.

These results show that for the 11/780 style of TB design, the process-space miss rate contains a significant irreducible component. Whereas we can expect steady enlargement of the system part of the TB to yield steady reduction of the system miss rate, we cannot expect any reduction in the rate of process-space fills. Indeed, the limiting value for the number of fills per interval as the TB grows is the average number of virtual pages touched in one context interval.

The number of fills could be reduced if process-space TB entries were tagged with a unique process ID number. A TB hit would then require matching both the virtual address and the requesting process' ID. Context switches would not need to flush anything from the TB. No such ID number exists in the VAX architecture, however, and we have not explored this design alternative.

Table XIV. Average TB Invalidate Headway (Instructions)

	TBIS	TBIA
μPC-TS1	4,110	8,730,000
μPC-TS2	10,190	2,410,000
μPC-EDU	3,410	934,000
μPC-SCI	2,960	[a]
μPC-COM	5,330	[a]

[a] Never occurred.

6.2 TB Invalidations

The μPC histogram measurements allowed us to measure two other events that flush TB entries. These were two special VAX instructions: TB Invalidate Single (TBIS) and TB Invalidate All (TBIA). TBIS is used by VMS when a single page-table entry is changed and it is necessary to make sure that the TB does not maintain a stale translation. TBIA is used by VMS to clear the entire TB when massive changes have been made to the page tables and it is easier to clear the entire TB than to figure out which pages to invalidate.

Table XIV shows the average instruction headway between executions of TBIS and TBIA. The average headway between executions of TBIS is in about the same range as the context-switch headway. If we assume that every TBIS caused one additional miss (the worst-case assumption), then we would expect one or two of these per context interval. TBIA is so infrequent that its effects are negligible. TBIS and TBIA, then, do not affect TB performance very much.

6.3 PC Changes

As was reported above, most I-stream TB references are due to branch instructions. Branches are very common, a property the VAX architecture shares with many others (see [15, 16]). Table XV shows that roughly one quarter of all VAX instructions executed in all our experiments changed the PC. This number includes explicit branches, both conditional and unconditional, as well as subroutine and procedure calls. (The actual measurements in the Dynaprobe and μPC experiments counted instruction buffer flushes; these occur very slightly more often than once every branch instruction, since they occur on traps, faults, interrupts, and resumptions of interruptable instructions.)

The Dynaprobe and simulation experiments allowed the branches to be classified according to whether the destination instruction was in process or system space. The results are shown in Table XV. All of the Dynaprobe runs and most of the simulation runs showed a much higher branch frequency in process space than in system space. In fact, according to the Dynaprobe results, an instruction in process space is about twice as likely to branch as one in system space.

Why should this be so? Here are three possible explanations:

(1) Most system code was written in VAX assembly language, and most process

Table XV. Instructions That Change the PC (Percent)

	Process space	System space	Total
Dyn-DAY1	31	16	28
Dyn-DAY2	33	14	30
Dyn-DAY3	28	19	27
Dyn-EDU1	30	14	25
Dyn-EDU2	30	13	26
μPC-TS1	—	—	27
μPC-TS2	—	—	29
μPC-EDU	—	—	26
μPC-SCI	—	—	24
μPC-COM	—	—	27
Sim-DIR	34	22	23
Sim-FORT	31	21	30
Sim-LINK	23	25	24
Sim-MAIL	27	21	22
Sim-EDT	29	20	26
Sim-RNO	26	23	25
Sim-SORT	23	24	23

code in some high-level language. With this greater degree of control over the code, VMS programmers avoided branches where possible.

(2) VMS programmers coded conditional branches so that they were more likely to fail (fall through) than succeed. High-level language programs cannot generally do this.

(3) It was hypothesized in Section 4.1 that system data structures were more complicated than process structures, and that system loops were iterated less than process loops. These characteristics could yield code with fewer branches.

Since conditional branches that fail to branch are not counted as PC-changing instructions, the dynamic frequency of instructions that *might* change the PC is even higher than shown in Table XV.

7. VARIATIONS OF THE TB STRUCTURE

7.1 Hardware Measurement Of TB Configuration Alternatives

By setting some special control bits at system boot time, one can reconfigure the 11/780's TB from 128-entry, 2-way associative to 64-entry, direct-mapped. This preserves the process/system split, with 32 entries for each. We did this in the Dynaprobe setting and collected one set of data for an additional DAY and one set for an additional segment of the EDU workload. Table XVI shows the results.

The miss rate increased by a factor of about 2 to 3 with the smaller TB, as did the percentage of total cycles spent in miss service. But process and system misses changed very differently. The size reduction had a much greater effect on process-space misses, which were increased by a factor of 3 to 5, than it did on system misses, which were increased by around a factor of 2. The context-switch

Table XVI. Effects of Hardware Reduction of the TB

	DAY		EDU	
	Full TB[a]	Half TB	Full TB[a]	Half TB
Process space misses per 100 instructions	0.93	4.52	1.00	3.33
System space misses per 100 instructions	1.65	3.76	3.28	5.72
Total misses per 100 instructions	2.58	8.28	4.28	9.05
Time spent in miss service (percent of cycles)	5.5	14.5	8.0	14.5
VAX MIPS	0.494	0.431	0.420	0.382
Relative performance	1.00	0.87	1.00	0.91

[a] Full TB numbers are unweighted averages.

flushes are a likely culprit: in Section 6 we showed that for three of the simulations, 32 entries would not be enough to accommodate all of the fills in the average context interval. Certainly 32 entries are too few for the system as well, but explicit full flushes of the system half of the TB do not occur, so the degradation in TB performance was not as pronounced as for process space.

7.2 Simulation Of TB Configuration Alternatives

The 11/780 hardware provides only a limited capability to vary the configuration of the TB. Simulation was necessary to evaluate TB configurations beyond those alternatives. Using simulation, we investigated the effect of size and associativity on miss rate, and the effect of separating the TB into process and system halves. In general, we have explored the design space around the 11/780.

The first parameters to be investigated were the size and associativity of the TB. Figure 6 shows TB miss rate versus TB size while holding the associativity constant at 2-way. It shows results for each of the simulated programs, and the solid dot represents the 11/780 measurement reported in Section 7.1. The figure shows that doubling the size of the 11/780 TB will result in an average savings of approximately 1.1 misses per 100 instructions. Using the results from Section 5, this would correspond to a 2.3 percent decrease in execution time for these benchmarks. Doubling the size again results in a further savings of 0.4 misses per 100 instructions, for a total reduction in execution time of 3.3 percent.

Figure 7 shows TB performance for the same set of TB sizes with a 1-way associative or direct-mapped configuration. Note that in both these charts, size is measured in number of TB slots, which is one of the principal determinants of the cost of the TB. Again the solid dot is used to represent the 11/780 measurement reported in Section 7.1. Comparison of Figures 6 and 7 shows that the reduction in miss rate gained by taking a direct-mapped TB and making it 2-way associative is almost equivalent to doubling its size.

Some details of the miss rates in the EDT benchmark are illustrated in Figure 8. The EDT benchmark was selected for illustration because its miss rate closely

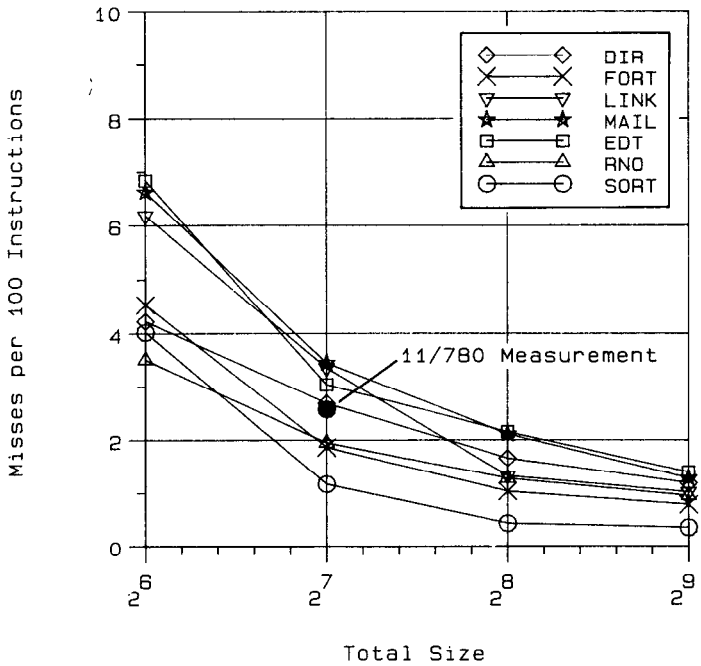

Fig. 6. TB miss rate versus size (2-way associative).

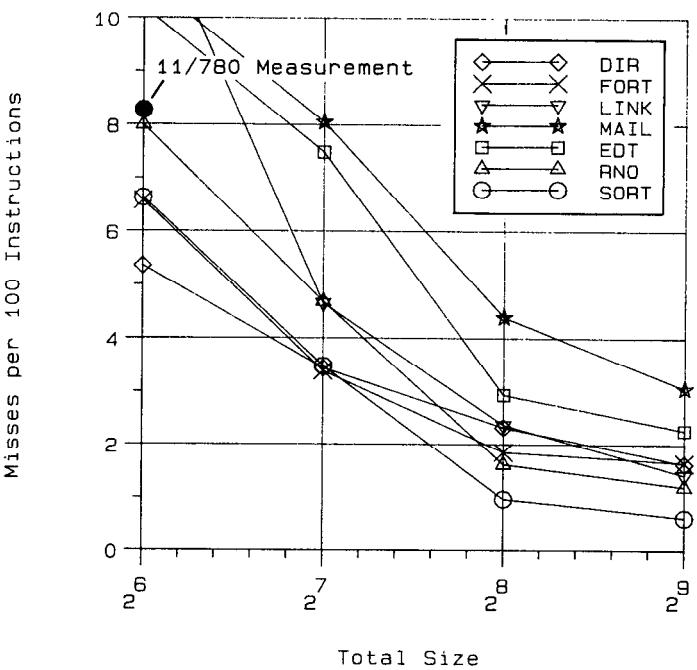

Fig. 7. TB miss rate versus size (1-way associative).

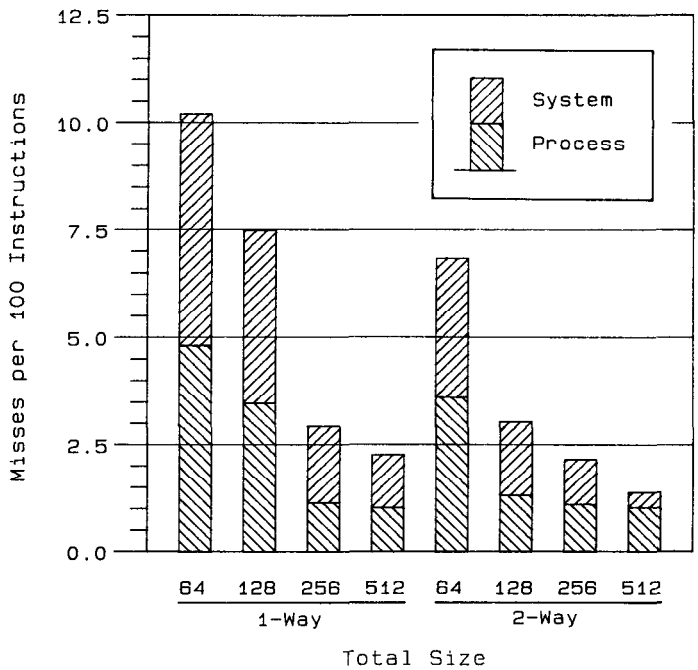

Fig. 8. TB miss rate (EDT benchmark).

resembles the aggregate miss rates measured with the Dynaprobe and μPC monitors. The bar chart illustrates the effect on miss rate as the size of the TB is increased from 64 to 512 tag entries and for both 1-way and 2-way associativity. Furthermore, the total miss rate is shown as being composed of two components: a process-space miss component, which is the bottom part of each bar, and a system-space miss component, which is the top part of each bar.

The figure shows the steady decline in the system-space component of the miss rate as the size of the TB increases. This behavior is to be expected, since the system half of a larger TB will allow more of the program's system-space translations to be resident in the TB, resulting in less contention for individual tag locations. In contrast to this behavior is the asymptotic approach of the process-space component of the miss rate to a much larger value of approximately one miss per 100 instructions. This is due to the frequent flushes of the process-space half of the TB. Even with large TB sizes there remains a significant number of *fills* following each flush (see Section 6.1).

Is there an overall advantage to having a separate system TB? Let a *split* TB be one with separate system and process halves, and let a *joint* TB be one in which process and system entries are combined. As was noted earlier, the advantage of a split TB is that system-space translations need not be flushed on process context switches. We assume that a joint TB, on the other hand, must be completely flushed on a process context switch. (An alternative, feasible in VLSI, would be to flush from a joint TB only the process-space entries.)

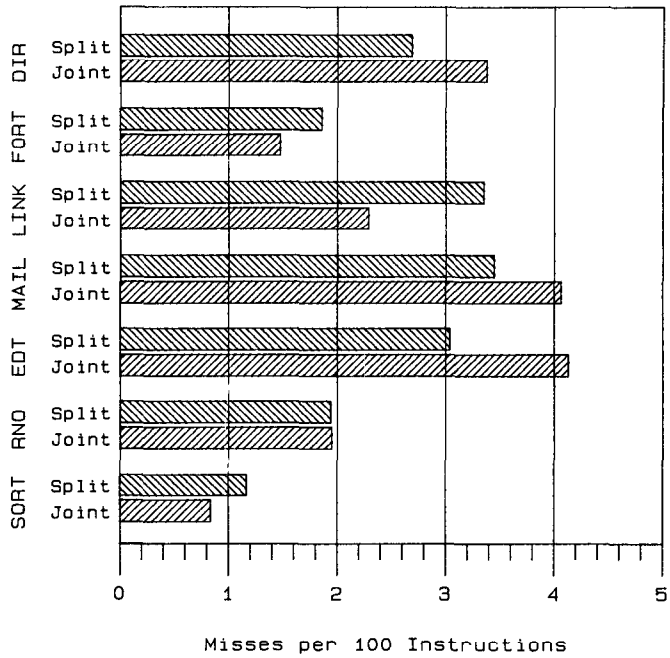

Fig. 9. TB miss rate for split and joint TBs.

Figure 9 illustrates the split versus joint comparison for TBs otherwise configured like the 11/780's. The figure indicates that for this set of benchmarks a split TB is not universally superior. Examination of those programs for which the joint TB was better (FORT, LINK, and SORT) revealed them to be those with miss rates larger for process space than for system space.

In general, a joint TB helps reduce a high process- or system-space miss rate by giving each address space additional TB entries. The three programs noted above benefited from the additional process-space entries. However, a joint TB will not be as effective at reducing a high system-space miss rate, because it must also offset the cost of flushing system entries on context switches. Furthermore, for either address space, the additional TB entries are bought at the expense of sharing the entries with the other space. This sharing exposes one space's entries to being bumped by misses in the other space.

Since the benefit of a joint TB is essentially manifest as an apparent increase in size, and simply increasing TB size ceases to be effective as the TB gets too large, one expects that for very small TBs a joint TB will be superior. Conversely, as the TB size increases, a split TB becomes superior. Table XVII illustrates this effect by listing the crossover sizes for the benchmarks simulated for both 1-way and 2-way associative TBs. The table also indicates that increasing associativity tends to reduce the size at which it is better to split the TB. This follows since increasing associativity makes the TB appear larger, thereby reducing the benefit of having a joint TB.

Table XVII. Least Size at Which Split TB Is Superior

	1-way	2-way
Sim-DIR	<64	<64
Sim-FORT	256	512
Sim-LINK	256	256
Sim-MAIL	<64	<64
Sim-EDT	256	128
Sim-RNO	256	128
Sim-SORT	512	256

8. CONCLUSION

We used three different techniques to study and evaluate translation buffer performance in the VAX-11/780. The Dynaprobe technique allowed direct access to hardware signals and was thus the best of the three for evaluating exactly what the hardware was doing. Its major drawbacks were that necessary hardware signals were not always conveniently available on the backpanel, and that *all* signals of interest had to be specified by the experimenter beforehand. The micro-PC histogram technique was more restricted in scope, since it could measure only phenomena expressed in the microprogram PC. However, it did allow the experimenter to measure new quantities after the fact. Simulation was the most flexible technique, allowing us to look at variations of the TB configuration as described in Section 7, but only gross performance characteristics could be seen. Timing information, in particular, was not available.

The simulation data show that selected single programs do have TB behavior similar to the aggregate behavior of real time-sharing systems. The data also emphasize the importance of choosing a program with significant system activity. The programs with the best simulated TB performance (FORT, RNO, and SORT) were also the ones with the fewest system-space misses and were not good representatives of the time-sharing systems. (We note in passing that FORTRAN compilers are frequent subjects of simulation experiments.) The editor EDT turned out to be a fair representative of the larger experiments when its system references were counted, but if only process misses and process instructions were counted, its miss rate per 100 instructions would decrease from 3.0 to 2.1.

Our measurements provided a good characterization of the impact of the TB on instruction performance. The product of the miss rate per instruction and the average service time per miss gives the time cost of the TB per average VAX instruction. We observed a typical miss rate of about 3 per 100 instructions and a service time of about 22 cycles per miss. This yields a cost of around 0.7 cycles out of the 10 spent in the average instruction.

The 11/780 was the first VAX, and thus its TB design choices were made without the benefit of extensive VAX simulation and measurement results such as ours. In view of this, our results, generally speaking, validate VAX architectural choices and 11/780 implementation choices concerning the TB. Its size and associativity seem about right (Section 7.2); the allocation of process page tables

in virtual memory does not produce many double misses (Section 4.3); and the microcode implementation of the TB service routines has acceptable performance cost (Section 5).

The results do not settle the question of split versus joint TB. Some of the simulated programs (at least the portions captured in our traces) would perform slightly better if the 11/780 TB did not differentiate system from process entries. As we discussed in Section 7, the joint organization is better for small TBs, and the split organization is better for large TBs; the 11/780's TB size seems to be near the boundary between small and large in this sense.

There is some room for performance enhancement of the 11/780 TB. It could be made bigger: a fourfold increase in size might cut the miss rate by a little more than half (Figure 6). A very much bigger TB could hold *all* of the system-space page mappings, but flushes due to process context switches prevent a like elimination of process misses. An architectural change that could help in this regard would be to tag each process TB entry with a unique process ID. This would eliminate flushing, but of course, unless the TB were quite big, context switching might accomplish *de facto* flushes by gradually replacing one process' tagged entries with another's. One final place where performance could be improved would be through the addition of hardware assists in the TB service routine. Cutting the 22 cycles we measured in half, for example, would speed up the 11/780 by around 3 percent. The benefit of any of these changes must, of course, be balanced against the corresponding cost, both in added logic and in potential increases in cycle time.

ACKNOWLEDGMENTS

The Dynaprobe experiments were done jointly with J. J. Grady. Bob Stewart, the designer of the 11/780 TB, helped us determine where to put our measurement probes and helped interpret the results. Garth Wiebe and Jean Hsiao developed the μPC histogram system. Dennis Ting developed the multimode trace facility, and Dave Orbits helped develop the simulator. Helpful comments on earlier drafts of this paper were offered by Dileep Bhandarkar, Wayne Cardoza, Dick Flower, Israel Gat, Kevin Koch, Jud Leonard, Matt Reilly, Alan Smith, Bob Stewart, Bob Supnik, Mark Truhlar, and Deborra Zukowski.

REFERENCES

1. ALPERT, D., CARBERRY, D., YAMAMURA, M., CHOW, Y., AND MAK, P. 32-bit processor chip integrates major system functions. *Electronics 56*, 14 (July 14, 1983), 113–119.
2. CASE, R. P., AND PADEGS, A. Architecture of the IBM System/370. *Commun. ACM 21*, 1 (Jan. 1978), 73–96.
3. CLARK, D. W. Cache performance in the VAX-11/780. *ACM Trans. Comput. Syst. 1*, 1 (Feb. 1983), 24–37.
4. DENNING, P. J. On modeling program behavior. In *Proceedings of the Spring Joint Computer Conference, Volume 40*. AFIPS Press, Arlington, Va., 1972, pp. 937–944.
5. DIGITAL EQUIPMENT CORP. TB/Cache/SBI Control technical description—VAX-11/780 implementation. Doc. No. EK-MM780-TD-001, Digital Equipment Corp., Maynard, Mass., Apr. 1978.
6. DIGITAL EQUIPMENT CORP. VAX/VMS internals and data structures. Doc. No. AA-K785A-TE, Digital Equipment Corp., Maynard, Mass., 1981.

7. DIGITAL EQUIPMENT CORP. VAX-11 architecture reference manual. Doc. No. EK-VAXAR-RM-001, Digital Equipment Corp., Maynard, Mass., May 1982.
8. EMER, J. S., and CLARK, D. W. A characterization of processor performance in the VAX-11/780. In *Proceedings of the 11th Annual Symposium on Computer Architecture* (Ann Arbor, Mich., June 5–7). IEEE, New York, 1984, pp. 301–310.
9 GREENBAUM, H. J. A simulator of multiple interactive users to drive a time-shared computer system. Master's thesis, Project MAC Rep. MAR-TR-54, MIT, Cambridge, Mass., Oct. 1968.
10. INTEL CORP. Intel iAPX 432 general data processor architecture reference manual, preliminary ed. Intel Corp., Aloha, Oreg., 1981.
11. JAIN, R. K. Workload characterization using image accounting. In *Proceedings of the Computer Performance Evaluation Users Group 18th Meeting* (Washington, D.C., Oct.). National Bureau of Standards, Washington, D.C., 1982, pp. 111–120.
12. KAPLAN, K. R., and WINDER, R. O. Cache-based computer systems. *IEEE Computer 6,* 3 (Mar. 1973), 30–36.
13. LEVY, H. M., and ECKHOUSE, R. H. *Computer Programming and Architecture: The VAX-11.* Digital Press, Bedford, Mass., 1980.
14. LEVY, H. M., and LIPMAN, P. H. Virtual memory management in the VAX/VMS operating system. *IEEE Computer 15,* 3 (Mar. 1982), 35–41.
15. MCDANIEL, G. An analysis of a mesa instruction set using dynamic instruction frequencies. In *Symposium on Architectural Support for Programming Languages and Operating Systems* (Palo Alto, Calif., Mar. 1–3). ACM, New York, 1982, pp. 167–176.
16. PEUTO, B. L., AND SHUSTEK, L. J. An instruction timing model of CPU performance. In *Proceedings of the 4th Annual Symposium on Computer Architecture,* (Silver Spring, Md., Mar. 23–25). IEEE, New York, 1977, pp. 165–178.
17. SATYANARAYANAN, M., and BHANDARKAR, D. Design trade-offs in VAX-11 translation buffer organization. *IEEE Computer 14,* 12 (Dec. 1981), 103–111.
18. SCHROEDER, M. D. Performance of the GE-645 associative memory while Multics is in operation. In *Proceedings of the ACM SIGOPS Workshop on System Performance Evaluation* (Cambridge, Mass., Apr.). ACM, New York, 1971, pp. 227–245.
19. SMITH, A. J. Cache memories. *ACM Comput. Surv. 14,* 3 (Sept. 1982), 473–530.
20. STRECKER, W. D. VAX-11/780—A virtual address extension for the PDP-11 family computers. In *Proceedings of the National Computer Conference, vol. 47,* AFIPS Press, Reston, Va., 1978.
21. WATKINS, S. W. and ABRAMS, M. D. Survey of remote terminal emulators. NBS Special Pub. 500-4, Apr. 1977.
22. WIECEK, C. A. A case study of VAX-11 instruction set usage for compiler execution. In *Symposium on Architectural Support for Programming Languages and Operating Systems* (Palo Alto, Calif., Mar. 1–3). ACM, New York, 1982, pp. 177–184.

Received January 1984; revised 7 November 1984; accepted 16 November 1984