Performance of the GE-645 Associative Memory While Multics is in Operation

by Michael D. Schroeder *

ABSTRACT

The Multiplexed Information and Computing Service (Multics) of Project MAC at M.I.T. runs on a General Electric 645 computer system. The processors of this hardware system contain logic for both paging and segmentation of addressable memory. They directly accept two-part addresses of the form (segment number, word number) which they translate into absolute memory addresses through a series of indexed table lookups. To speed this address translation each processor contains a small, fast associative memory which remembers the most recently used address translation table entries. This paper reports the results of performance measurements on this associative memory. The measurements were made by attaching an electronic counter directly to a processor while Multics was in operation, and were taken for several associative memory sizes. The measurements show that for the observed load 16 associative registers are enough.

---

* Massachusetts Institute of Technology, Project MAC, Cambridge, Massachusetts. Work reported herein was supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Project Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

Introduction

The Multiplexed Information and Computing Service (Multics) of Project MAC at M.I.T. [1] incorporates both paging and segmentation. The information stored in the system is organized as a collection of segments. Memory for segments is allocated in fixed-sized blocks which are paged in a multi-level memory system. Segmentation allows controlled sharing of information and provides considerable help to a program in organizing storage, even at the level where the program is expressed in machine language. Paging simplifies the problem of physical storage allocation in a time-shared computer system. (For a general discussion of paging and segmentation see [2].)

The use of paging and segmentation by Multics is made possible and practical by the hardware on which Multics operates, the General Electric 645 computer system. The 645 central processing units [3] are designed to allow paging and segmentation of addressable memory (core memory in this case). The algorithm implemented by the addressing logic of these processors applies a segmented, paged structure to the core memory. This addressing logic accepts two-dimensional (segment number, word number) addresses which it translates to the corresponding absolute core addresses. The translation algorithm involves a series of indexed lookups in address translation tables in core.

Each processor contains a small, fast associative memory which remembers the address translation table entries most recently used in address translation. The associative memory

exploits the premise that most programs exhibit sufficient locality of reference to cause consecutive reuse of a small set of address translation table entries. Under these conditions the associative memory allows most two-dimensional addresses to be translated to absolute form without reference to the tables in core.

This paper reports the results of a series of experiments performed on the associative memory of a GE-645 processor to verify the locality of reference premise. An electronic counter was attached to the logic of a processor while Multics was in operation and a normal user load was present. The counter recorded the occurrence of various events associated with processor and associative memory operation. The measurements allow various performance parameters of the associative memory to be calculated, and the validity of the locality of reference premise to be determined. The same set of measurements were taken for associative memory sizes of 16, 8, and 4 registers, and with the associative memory turned off.

The paper is presented in four parts. First the segmented, paged structure applied to core memory is described and the address translation algorithm used by a processor is specified. Next the role of the associative memory in address translation is presented. The third section describes the measurement experiments and presents the results obtained. In the last section the significance of these results is discussed.

Address translation

The addressing logic of the GE-645 processors functions at two levels and solves two distinct problems. At the first level is segmentation. A machine language program for a GE-645 processor executes in a two-dimensional address space. The address space is a collection of independent segments identified by number. Each segment is an arbitrary length array of 36-bit words. An address of the form (s,w) identifies the wth word of the segment numbered s.

The collection of segments in the address space is defined by a descriptor segment. The descriptor segment contains an array of segment descriptor words (SDWs), each of which describes a single segment in the address space. The number of a segment is just the index of the corresponding SDW in the descriptor segment. Among other things, an SDW contains the absolute address of the beginning of the corresponding segment in core. The absolute address of the beginning of the descriptor segment is contained in the descriptor base register (DBR) of a processor.

Address translation in this (simplified) environment is straightforward. To determine the absolute core address corresponding to the two-dimensional address (s,w) a two step calculation is made. First, the segment number s is added to the address in the DBR to calculate the absolute address of the SDW corresponding to the segment. Then, the word number w is added to the address contained in this SDW to calculate the absolute address corresponding to the address space location (s,w). The

addressing logic of the processors performs this calculation each time a two-dimensional address is used.

At the second level of function is paging. Paging complicates the simplified address translation algorithm just sketched. Storage for segments in core, including the descriptor segment, is not a contiguous array of locations as the simplified algorithm implies. Rather, storage for segments is allocated in scattered 1024-word blocks. A page table, also in core, lists in an array of page table words (PTWs) the blocks of core occupied by the consecutive pages of a particular segment. This paging of segments is transparent at the machine language level, but is taken into account by the address translation algorithm in converting a two-dimensional address to absolute form. The absolute address in the DBR of a processor is actually the address of the beginning of the page table for the descriptor segment. The absolute addresses in SDWs are the beginning locations for the page tables of the segments in the address space.

The address translation algorithm used by the GE-645 processors, then, takes into account both segmentation and paging. Figure 1 illustrates this algorithm. The steps in translating the address (s,w) are shown. The absolute addresses in the DBR, the PTWs, and the SDWs are represented by arrows. It is assumed that all information is in core, as only this primary level of memory is directly accessible to the processors. Page tables and pages not in core would be indicated by fault flags in the appropriate SDWs and PTWs, respectively. The addressing

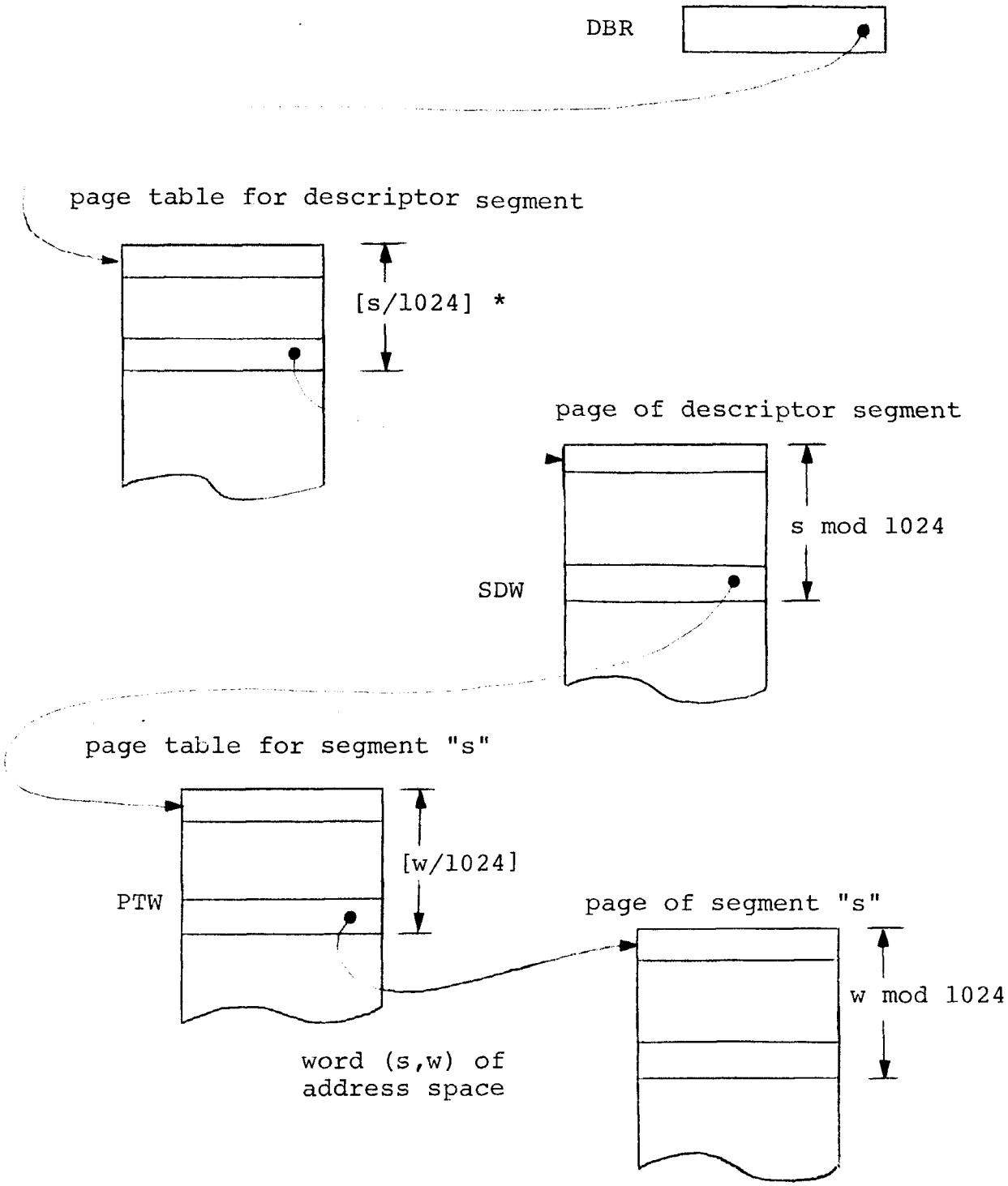

Figure 1: Translation of address (s,w) by GE-645 processor

---

* [...] indicates "integer part of"

logic of the processors "traps" when such a fault flag is encountered during address translation.

Note that simply changing the value in the DBR of a processor will cause the address translation algorithm to interpret two-dimensional addresses relative to a different set of address translation tables. In fact, this property is used by Multics to provide a separate two-dimensional address space for each on-going computation. (These address spaces may overlap.) An on-going computation together with its address space is called a process. The address space of a typical Multics process contains about 200 segments. (For a more complete description of segmentation and paging in Multics, and the addressing logic of the GE-645 processor, see [4]).

Role of associative memory

The address translation algorithm described above implies that reference to a word in the address space of a process requires three core references for address translation in addition to the requested address space reference. In order to reduce this ratio of four absolute core references per address space reference, each GE-645 processor contains a 16 register associative memory. This fast memory retains the 16 most recently used SDWs and/or PTWs. (PTWs for the descriptor segment are an exception, and are never placed in the associative memory.) Under conditions of consecutive reuse of a small set of PTWs and SDWs, the associative memory can eliminate some of the core references for PTWs and SDWs that would otherwise

occur.

Each time the addressing logic of a processor is activated to translate a (segment number, word number) address, it interrogates the associative memory before referencing any SDWs or PTWs in core. The search key is the segment number and word number from the address being translated. Three results of this interrogation are possible:

1. PTW found
2. SDW found
3. no match

In case 1 the associative memory contained the PTW for the particular page of the segment being referenced. No further address translation references are needed. Adding the word number modulo 1024 to the absolute address contained in the found PTW yields the absolute address corresponding to the address space location being referenced.

In case 2 the associative memory did not contain the needed PTW, but did contain the SDW for the segment being referenced. Address translation may be completed with one PTW reference to core.

In case 3 nothing of assistance was in the associative memory. All three address translation references to core must be made.

The SDWs and PTWs in the associative memory are ordered by recentness of use. Each time an SDW or a PTW from the associative memory is used it goes to the head of the use queue. Whenever an SDW or a PTW is referenced in core because it was not

in the associative memory, it is placed in the associative memory. It replaces the item at the tail of the use queue and then goes to the head of the queue. Thus, the replacement algorithm used in the associative memory is strictly "least recently used".

The processor hardware performs the search of the associative memory in 200 to 600 nanoseconds (depending on the result of the search). By comparison, a core reference for an address translation table entry (an SDW or a PTW) requires 1.2 microseconds. The associative memory search is slow by 1971 standards because of the discrete component logic used.

The information in the associative memory of a processor is a copy of information that is also in core. The associative memory thus generates a "multiple copy" problem, and some way is required to keep the potential multiple copies of SDWs and PTWs identical. For this purpose there is a special "clear associative memory" feature on the processors. It is invoked by program whenever an SDW or a PTW in core is changed. If a multi-processor configuration is in use, the associative memory of each processor must be cleared. This feature is also invoked when the absolute address in the DBR of a processor is changed to cause the processor to execute in a different address space. When such an address space change occurs the associative memory will contain SDWs and PTWs for the previous address space which could cause errors if left.

Performance measurements

By attaching an electronic counter to a GE-645 processor while Multics was running, the effectiveness of the associative memory has been measured. A general problem with any large, complex computer system is the difficulty of predicting performance in advance of actual operation. The problem is that there is no convincing way to simulate the demand placed on the system by an actual user load. This was true of Multics in general, and of the associative memory of a 645 processor in particular. Although a simulation model for the associative memory was developed and exercised, its authors concluded that little confidence could be placed in the results produced. They reported that:

> In general, as the proportion of random references increases, and as the total number of relevant pages and segments increases, the number of associative memory cells required for "near optimum" performance also increases. Therefore, the results given ... should not be assumed to indicate overall auxiliary memory performance. [5]

In other words, there was no convincing way to model the sequence of address space references that would be generated by a Multics-like system with an actual user load. Curiosity about the relation of the number of associative registers to actual system performance could only be satisfied by experiments on an actual system. The experiments were to test the judgement implicit in the processor design that 16 associative registers are the right number.

The method was to isolate points in the processor logic where pulses corresponding to events of interest could be

detected by an electronic counter. The four events chosen were instruction executions, associative memory searches, "no match" associative memory search responses, and absolute core references by the processor. A series of counts for 10 second periods was made for each event. During the time the counts were being made the processor was part of the hardware configuration running Multics for a normal user load. This configuration included one processor and 256,000 words of core memory.

The same measurements were taken for associative memory sizes of 16, 8, and 4 registers, and with the associative memory turned off. (Users began to complain only in the last case.) The size of the associative memory was varied by making temporary modifications to the processor logic while the processor was not in service. The measurements were made in June, 1970 over the time of several days. During the measurements the number of simultaneous users of the system varied from 6 to 35. Multics limits the number of simultaneous users to maintain adequate response time. This limit has continuously risen during the development of Multics. At the time of the measurements the limit was 35. The user load during the measurements included approximately equal numbers of system programmers, application programmers (mainly using PL/1), and students. At least 25 counts were made for each event.

An electronic counter is an ideal instrument for such an experiment, because it does not influence the system being measured. Very elaborate counter-like devices with data recording capabilities have been constructed to perform similar

measurements on computer systems (for example, see [6]), but for this experiment a simple digital frequency meter with a seven digit display was used. For each count the meter was directly connected to some pin in the logic of the processor.

Figure 2 presents a summary of the results obtained by these experiments. The figure is largely self-explanatory. The "range" values are the percentage difference between the average of all counts for an event, and the maximum and minimum count remaining after deleting the lowest and highest quarter of the counts, respectively.

The widest variation of results occurred for the three "no match" events. This variation is caused by the short-term fluctuations in the computation load which are characteristic of an interactive system. Under Multics, when the processor is not busy performing user and system computations it executes in an "idle loop" waiting for more work. The "idle loop" is very small, and references only a few pages of code and data. Thus, while in the "idle loop", "no match" responses to associative memory searches will occur less frequently than otherwise. The number of "no match" responses per second will vary with the proportion of time the processor is executing in the "idle loop", and thus will vary with user activity. The variation in the number of users during the measurements amplified this effect.

Figure 3 presents the data of Figure 2 graphically. It is obvious from Figure 3 that the size of the associative memory has a direct effect on the instruction execution rate of the processor. All of the decrease in this rate that occurs as the

	Measurement	Average observed per second rate	Range
with 16 associative registers	Instruction executions	341,945	+2% -2%
	AM search requests	414,688	+2% -4%
	AM "no match" responses	5,180	+13% -5%
	absolute core references	419,425	less than 1% both ways
with 8 associative registers	Instruction executions	322,912	+2% -1%
	AM search requests	377,912	+1% -2%
	AM "no match" responses	11,002	+13% -6%
	absolute core references	418,284	+1% -2%
with 4 associative registers	Instruction executions	286,531	+2% -1%
	AM search requests	337,442	+1% -1%
	AM "no match" responses	35,710	+5% -6%
	absolute core references	451,895	+1% -1%
with no associative memory	Instruction executions	123,198	less than 1% both ways
	absolute core references	541,832	less than 1% both ways

Figure 2 : Summary of results of measurement experiments

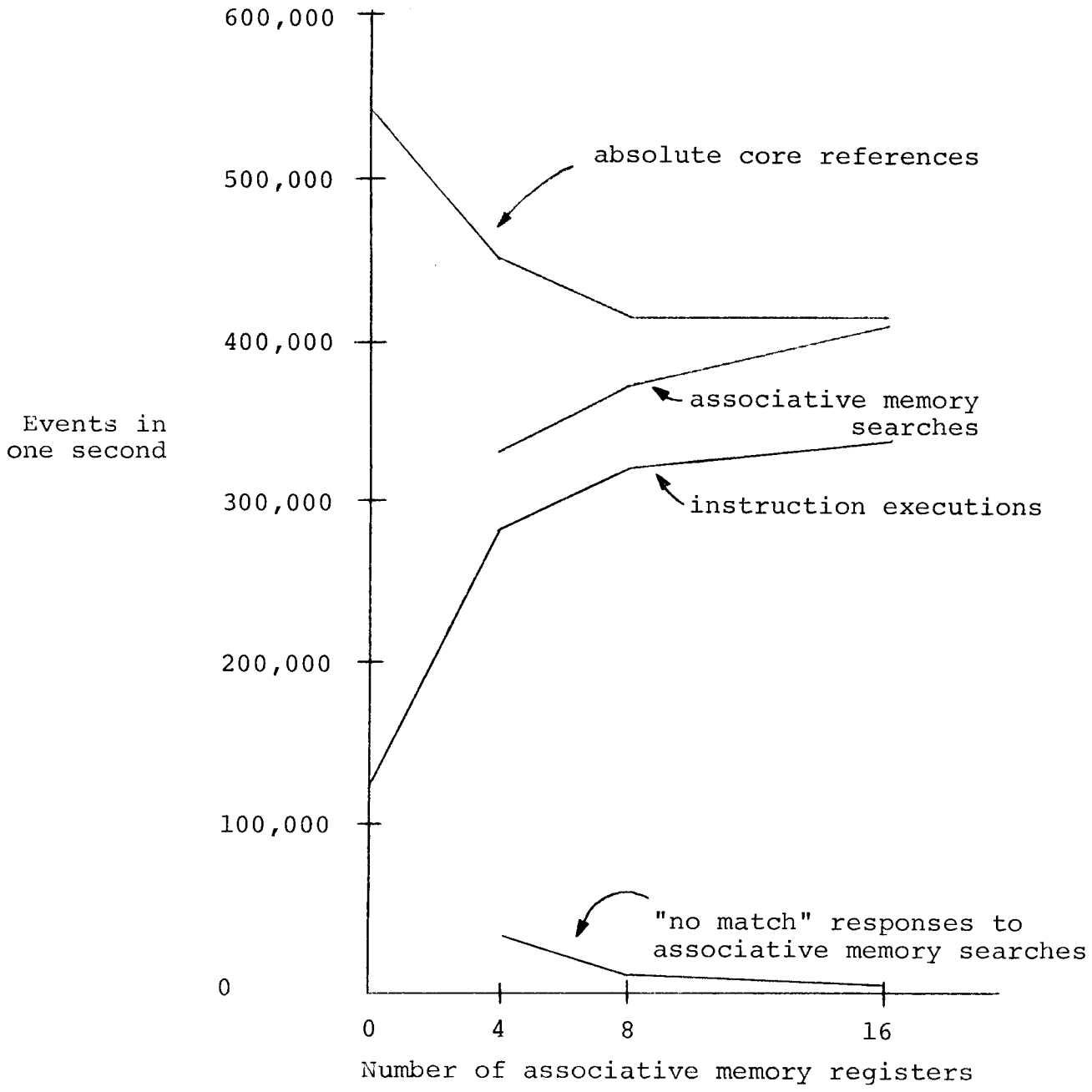

Figure 3: Performance with various associative memory sizes

size of the associative memory is decreased is caused by the retrieval from core of more and more SDWs and PTWs. Figure 4 illustrates this point quite well. The number of address space references (associative memory searches) per instruction execution remains relatively constant as the size of the associative memory decreases, while the number of absolute core references per instruction execution rises dramatically. That the number of address space references per instruction execution remains constant is a good check on the consistency of the various measurements.

Discussion of results

Two conclusions can be drawn from the results of these measurements. The first concerns the cost of segmentation and paging (block allocation) in terms of core references for address translation. Given that the decision to support segmentation and paging has been made, there are two obvious alternative implementations. One uses only internal processor registers for address translation information. The other places address translation information in core. The latter seems to be a less restrictive implementation and to require fewer hardware registers, but appears to cause core references for address translation not required by the former. The measurement experiments show that a small associative memory can reduce to a negligible number the core references for address translation required with the second implementation. With 16 associative registers on the GE-645 an average of approximately 5,000 "no

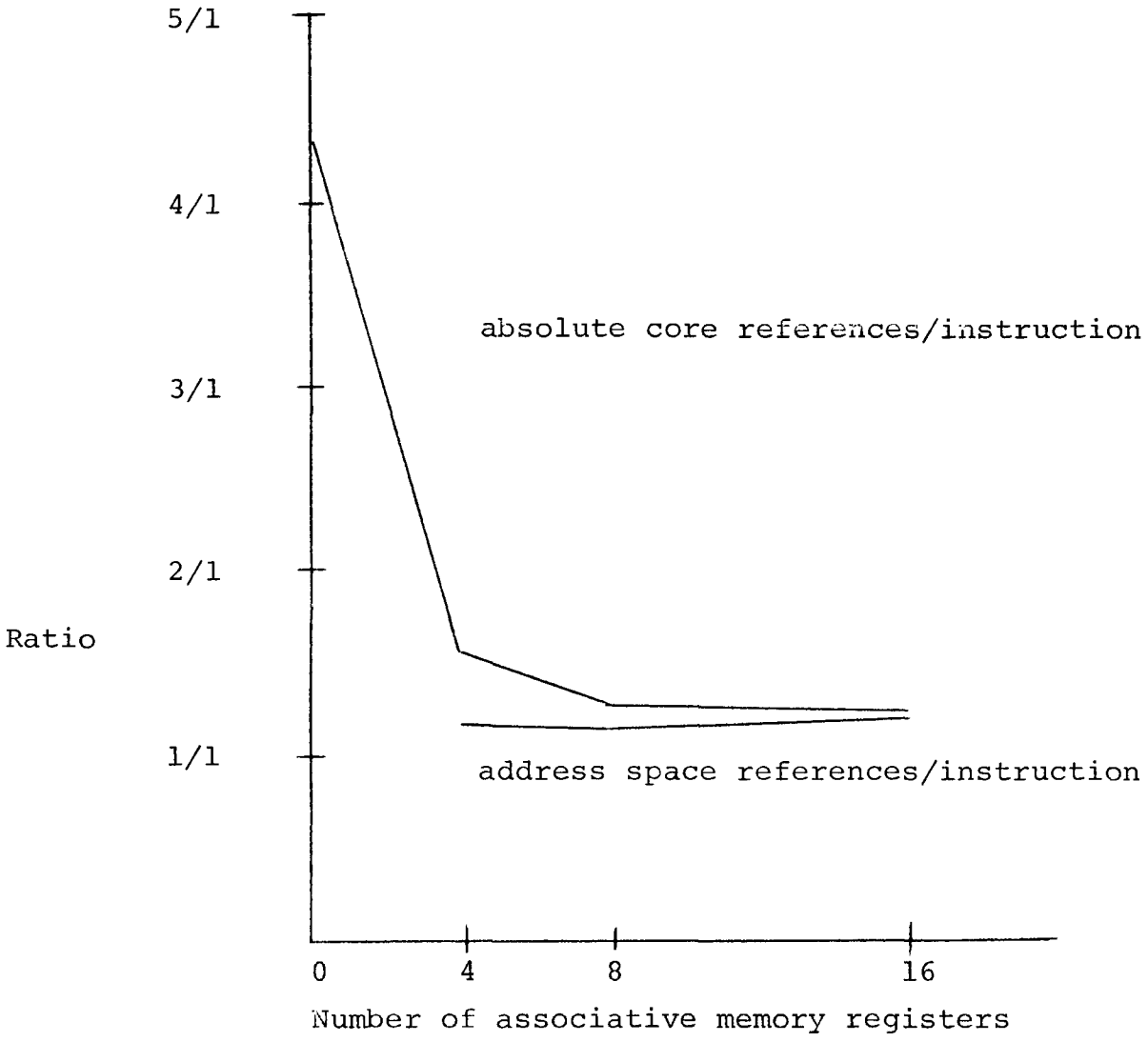

Figure 4: Core and address space references per instruction with various associative memory sizes

match" associative memory search responses were observed each second. Each of these generates at most three absolute core references. Thus, out of the approximately 420,000 absolute core references observed each second only 15,000 are attributable to address translation, or about 3.5%. Put another way, each address space reference required only 1.03 core references rather than the four implied by the address translation algorithm.

It is interesting to note that over half of the address translation references to core may be associated with the clearings of the associative memory discussed earlier. Independent measurements show that the associative memory is cleared about 170 times a second. Because up to 16 "no match" responses are generated by each associative memory clearing, up to 16*170*3 = 8160 of the "no matches" each second - over half of them - are attributable to the clearings of the associative memory.

The second conclusion is that 16 associative registers are enough for a 645 processor running Multics. At 16 registers the curve of the instruction execution rate with respect to the number of associative registers in Figure 3 has become quite flat. Furthermore, any increment in the number of associative registers above 16, no matter how large, could do no more than eliminate some fraction of the 3.5% of all core references attributable to address translation. Thus, the prospect of significantly increased performance with more associative registers seems remote. Apparently 16 is approximately the largest number of associative registers that can be effectively

utilized.

Whether 16 or a smaller number is the proper number of registers to include in the processor depends on the cost of associative registers. Reduction of the number of associative registers to 8 resulted in a 6% decrease in the instruction execution rate. This might be acceptable if associative registers were very expensive. A cost/performance tradeoff must be calculated in order to decide between 16 and 8 registers.

Acknowledgements

John Ammons of the General Electric Company's Cambridge Information Systems Laboratory (now part of Honeywell Information Systems, Inc) contributed much time and effort to performing the measurement experiments. Roger Schell of M.I.T. suggested that the paper be written. F.J. Corbató, R.P. Goldberg, and J.H. Saltzer, all from M.I.T., provided helpful suggestions on the content of the paper.

References

[1] Corbató, F.J., et al, "A New Remote-Accessed Man-Machine System", AFIPS Conference Proceedings 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 185-247.

[2] Denning, P.J., "Virtual Memory", Computing Surveys 2, 3 (September, 1970), pp. 153-189.

[3] GE-645 Processor Reference Manual, Cambridge Information Systems Laboratory, General Electric Company, August, 1970.

[4] Bensoussan, A., C.T. Clingen, and R.C. Daley, "The Multics Virtual Memory", Second ACM Symposium on Operating Systems Principles (October, 1969), Princeton University, pp. 30-42.

[5] Shemer, J.E., and G.A. Shippey, "Statistical Analysis of Paged and Segmented Computer Systems", IEEE Transactions on Electronic Computers EC-15, 6 (December, 1966), pp. 855-863.

[6] Schulman, F.D., "Hardware measurement device for IBM system/360 time sharing evaluation", Proceedings of 22nd National Conference of ACM (ACM Publication P-67), Thompson Book Company, Washington D.C., 1967, pp. 103-109.