AUGUST 1976

HEWLETT-PACKARD JOURNAL

Series II General-Purpose Computer Systems: Designed for Improved Throughput and Reliability

A larger, faster memory system with error correction and error logging, a faster central processor, an expanded instruction set, and a more efficient operating system are the major technological advances. Benchmark studies rate the new HP 3000 Series II Computer Systems at two to four times the throughput of earlier versions.

by Leonard E. Shar

LIKE EARLIER VERSIONS OF THE HP 3000 Computer System,[1] the new HP 3000 Series II is a virtual-memory, multilingual, multiprogramming computer system capable of performing batch operations and multiple-terminal on-line functions simultaneously. The Series II has the same basic architecture and input/output hardware as its predecessors, and software compatibility has been preserved. Virtually everything else is new.

To the user, the principal difference is in performance. Overall throughput has increased by a factor of two to four for a "typical" job mix, and some programs have run as much as ten times faster (see page 14). A larger main memory address space is the reason for most of the performance improvement, but there are also operating system enhancements, added instructions, firmware improvements, and some hardware changes.

Series II main memory is all semiconductor, based on 18-pin 4K RAM chips. An unusual feature is a new fault control system that detects and corrects memory errors with no reduction in speed. 3000 Series II machines automatically log each error corrected along with the identity of the component that caused it. Failing parts can be weeded out of the system to minimize future errors and assure continuous operation. Thus the Series II is expected to be much more reliable than earlier systems.

There are three compatible models in Series II, ranging from the basic Model 5 to the highest-performance Model 9 (Fig. 1). Model 7 is intermediate in performance and cost. All three models can compile and execute programs written in any or all of five languages: SPL, RPG, COBOL, BASIC, and FORTRAN.

About the HP 3000

To understand what's been done in the new Series II, it's helpful to have some knowledge of HP 3000 architecture. Here is a brief review.

There are two principal modes of operation of the CPU. "User" mode is one in which all operations performed are strictly checked by the hardware to ensure that only the user's own data may be accessed. Any code executed in user mode is completely safe in the sense that it cannot affect the operation of other users or of the Multiprogramming Executive operating system (MPE). The other mode, called "privileged", is reserved for the operating system only and bypasses all the checking normally performed by the

Cover: *Against a backdrop photo of an HP 3000 Series II Computer System are a Series II memory board and an example of an Error Correcting Memory Log Analysis. The Series II memory corrects all single-bit memory errors so they don't affect normal operation. Errors are logged so that failing 4K RAM chips can be weeded out of the system.*

In this Issue:

Fig. 1. *Model 9, the largest HP 3000 Series II system, has the capabilities of a small-to-medium full-scale general-purpose computer, but is much lower in cost. It can meet the needs of larger commerical or scientific users. Model 7, a smaller system, is especially appropriate for users of small business computers who want to move upwards to on-line data base management. Model 5, the smallest Series II system, is a low-cost terminal-oriented system that also does concurrent batch processing.*

hardware.

Memory is logically divided into variable-size segments either of code (which cannot be modified) or of data. The complete set of all such segments known to the system constitutes the virtual memory. All segments reside on disc storage until required in main memory. To execute code or access data, the relevant segment must be present in main memory (sometimes called real memory). Whenever a segment is referenced the hardware checks to see whether it is in main memory; if it is not, the operating system is invoked to bring it in. Thus the management of the virtual memory is totally automatic and transparent to the user, and the system can reference a virtual memory space far larger than the real memory available.

Each user has a data stack that resides in a data segment with a maximum size of 32,768 16-bit words. Only 15 bits of address are required to locate any one of these words. The use of all 16 bits for indirect addressing and the 16-bit index register facilitate addressing to the byte level. All addressing is relative to a set of registers set up automatically by the operating system for each user prior to execution. Fig. 2 shows this register structure.

Registers DL and Z delineate the area of data that the user may access. Direct access (indexed or not) can be relative to one of three other registers, DB, Q, or S. All indirect access is relative to DB, the data base register. The different addressing modes are a natural extention of the different types of variables used in a program, and are automatically chosen in the most convenient way by the compilers so the user need not know of their existence.

Two registers, PB and PL, delineate the particular code segment being executed by the user. A third register, P, is the program counter and may be considered to point to the next instruction to be executed, although this is not strictly true in general because of instruction look-ahead in the CPU. The hardware does not allow an instruction to be fetched outside the code segment, that is, outside the range PB to PL. The only way to access another code segment is to call a procedure in that segment. This is done via the PCAL instruction, which sets up new values in the PB, PL, and P registers. These values are derived from tables maintained by and accessible only to MPE.

A user cannot access anything outside the area of memory that MPE has set aside for him. Furthermore, every access to memory is addressed relative to one or more of the registers controlled solely by MPE. The user does have control over certain local data registers, such as the index register, as well as implicit control over the top-of-stack registers, which are invisible to the code. These local registers and the environment registers are saved automatically by the hardware when the user is interrupted. Therefore the user can be stopped at any time and his data and/or code areas can be moved to another location in memory without any effect on the user. Thus a user cannot find out about anything outside his areas (except what he can get from special carefully protected

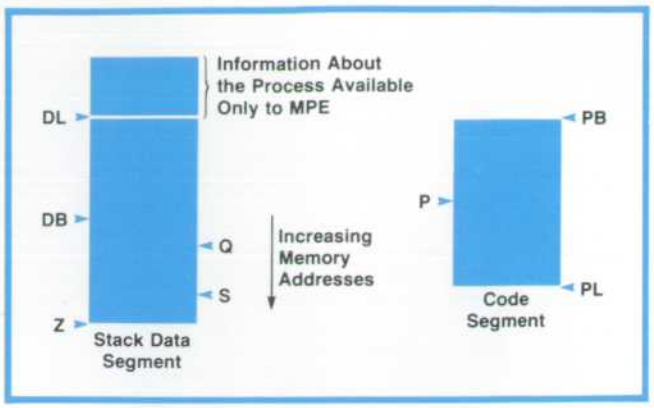

Fig. 2. *Basic HP 3000 register structure. Memory is divided into variable-size segments of code and data. DL, DB, Z, Q, S, PB, PL, and P are registers that delineate various areas of the code and data segments.*

procedures in MPE); in particular, he is unaware of his physical location in memory or even the size of real memory on his system. However, the size of real memory has a significant effect on overall throughput and the response time observed by the user.

Expanding Memory

In increasing the available real memory on the system, the major constraint was that all user code that previously ran on the HP 3000 would have to run on the expanded memory system; at most recompilation could be required in some cases. All current users would then be able to upgrade their systems without difficulty, and it would still be possible to use the large quantity of software already developed. To fulfill this requirement it was essential that all user mode instructions have the identical effect on the user's data on both machines; this precluded any change to the addressing modes allowed to the user. New instructions could be added but none could be changed.

This proved relatively easy to do because of the elegant structure of the HP 3000. Since a user is unaware of where in memory his program is executing it was a simple matter to add memory beyond the 64K words normally addressable by 16 bits. This was done by dividing main memory into four banks of up to 64K words each. Each memory location can be uniquely specified by a bank number and its address within that bank. So long as no code or data segment can cross bank boundaries, all addresses within each segment can be calculated in the normal way using as a base only the 16-bit address within the bank; after this calculation (and bounds check) the bank number is appended to the left of the address to provide the unique address of the required location. This extended address is used to select the location within the correct memory module. Since the user cannot modify the data or code base registers it was possible to extend these registers beyond 16 bits. The bank structure guarantees that for any legal address calculation there will never be any overflow out of the 16 low-order bits, so the user need not even know about the excess bits. The only instructions that cannot use this mechanism to access memory are certain privileged instructions that do absolute addressing. Since this access method is so consistent with the existing instruction set all user code remains valid. Only the operating system had to be modified since it is the only code that is aware of the existence of the larger memory and the bank number.

A set of new privileged instructions were added to allow MPE to access absolute memory beyond the previous 64K word limit. It was also necessary to change the existing privileged instructions that deal with registers inaccessible to the user. Furthermore, the operating system has another privileged mode of operation in which it is allowed to switch the DB register to point to some data segment other than the currently executing stack; for complete generality it is necessary to allow the stack and the extra data segment to reside in different banks. Three new bank registers in the CPU make it possible for the HP 3000 Series II to support any addressing mode, user or privileged, and at the same time allow any segment to be in any bank. The three bank registers are designated DB bank, stack bank, and PB bank (see Fig. 3).

A further constraint on the design of the extended memory HP 3000 was that as little peripheral hardware be changed as possible. To meet this objective it was desirable to be able to use all existing input/output interfaces. This also simplifies and reduces the cost of upgrading existing installations. However, it is essential that I/O be possible to or from any area in memory—if for no other reason than that the memory management system must be able to transfer segments between virtual and real memory. There are

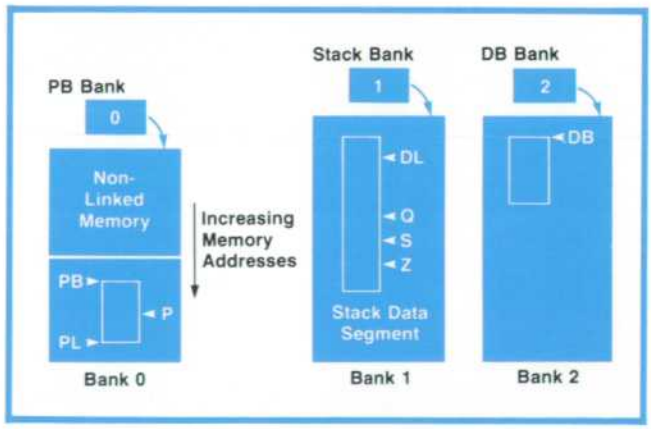

Fig. 3. *Modified register structure of the HP 3000 Series II. Memory is divided into four banks of up to 64K words each. Three bank registers (shown at top) allow code and data segments in different banks to be used simultaneously.*

three hardware modules on the HP 3000 that are involved with I/O to and from memory, and only these (the I/O processor, multiplexer channel and selector channel) had to be modified to allow access to the expanded memory. Once again, changes are minimized by ensuring that no segments can cross bank boundaries. Since all I/O transfers take place on a segment basis, it is only necessary to set up the correct bank at the beginning of the transfer and then request the transfer to take place in the normal manner. A new "set bank" instruction preceding the standard channel programs permits the standard device controllers to access the extended memory. This instruction is interpreted within the channel where the bank number is stored and is appended to the memory address of all transfers for that device.

Software Memory Management

MPE divides main memory into two areas. The first, fixed memory, contains only the tables and code that the operating system requires to be memory resident. These include the interrupt handlers, the memory manager, and the scheduler. The remainder of memory is designated linked memory, and contains all other code and data. User and operating system segments are brought into this area by the memory manager as they are required. The architecture allows most of the operating system, including the file system, the command interpreter, the spooler, and even much of the I/O system, to be shared by all users without being memory resident. In fact, only 8% of MPE code is required to be in fixed memory, and the total size of fixed memory on the Series II can be as low as 25K words. This is only a little larger than on previous HP 3000 systems, so the expansion of linked memory on the Series II is far greater than the fourfold expansion of real memory. Measurements have verified that the overall performance of the system in a multiprogramming environment is determined by how well linked memory is used.

The greatly enlarged linked memory presents an opportunity for the operating system to do a much better job at keeping the "right" segments in memory. Basically the memory manager's job is to attempt to maximize the probability that a segment will be in real memory when it is needed by a process. A process is the basic executable entity; it consists of a stack data segment (see Fig. 3) that contains the data local to that process, at least one code segment (possibly shared with other processes), and possibly some extra data segments. Note that the "user" described earlier is really just an instance of a process, and in fact the subsystems and even parts of the operating system itself are other instances of processes. Each process is essentially independent of every other.

The dispatcher is the module of MPE that schedules processes for execution. Each process has a dynamically changing priority number, and the dispatcher keeps a list of active processes (those requesting execution) ordered by priority. This is called the ready list. The dispatcher manipulates the priority number so a process gets service appropriate to its creation parameters. The basic scheduling algorithm is to attempt to run the highest-priority active process. If that process is not in memory the dispatcher requests the memory manager to make enough of that process's segments present in memory to allow it to continue.

As a process runs it may require another code or data segment. If the segment is not present in main memory the hardware traps out (a segment trap is said to have occurred) to the memory manager, which schedules a request to bring in that segment before that process is allowed to continue. While waiting for the completion of that transfer some other process in memory may be run. It is clear that a process will run best when all the segments it references are in main memory. However, all the segments for all the processes will not fit in main memory, and this extravagance is unnecessary anyway because most of the code in any program is executed infrequently. This is the well documented concept of locality.[2,3] Thus it is more efficient for the memory manager to bring in segments as they are required on an exception basis.

If the memory manager can ensure that the process has enough segments in main memory so that it segment faults infrequently then the process will run efficiently and the overhead for virtual memory will be low. This gives rise to the concept of a working set, which is the set of segments required to be in memory for a process to run well. The problem is to determine what that working set is for each process. Very often even the programmer cannot guess what it is likely to be because it changes dynamically during execution. MPE uses a "segment trap frequency" algorithm to determine which segments belong to each process's working set. This algorithm is highly efficient.[4] A working set list is maintained for each process and the size of this working set is expanded or contracted in an attempt to arrive at a constant interfault time for each process. This is in effect a negative feedback control mechanism. MPE keeps track of the interfault time very accurately on a per-process basis with the help of a special process timer built into the CPU.

When a process implicitly requests an extra segment the memory manager will bring that segment into main memory after it has found space for it. At this time an important decision must be made: should this process have its working set expanded to include this segment or should one of its older segments be removed from its working set? This decision is based on

the CPU time used by that process since the previous segment trap. The decision is important because if the working set is expanded too much this process will tend to control more than its share of main memory, thus degrading the performance of other processes. On the other hand, if the working set is too small the process itself will run inefficiently even though it has little effect on any other processes. We use a time between segment faults of 100 ms to assure both efficiency within each process and overall system efficiency.

When segments are to be removed from the working set they are merely delinked from the list associated with that process and linked onto a single system-wide overlay selection list. Although this list will be used later to find segments to be overlayed, at this point the segments remain in main memory. For a data segment an anticipatory write operation is initiated to update the virtual memory image of that segment. If there is more main memory on the system than is required to support all working sets of currently active processes, it is especially advantageous to leave the segments in main memory at this point. For example, if only one process is active it is conceivable that all the segments it has ever referenced will actually be in main memory (because no other process has requested any memory) even though only a small percentage of them will be in its working set. In this way a process can use main memory far in excess of its working set, but only to the extent that there is extra unused memory available at that time.

Another important memory manager decision is which segments to remove from main memory (i.e., overlay) when space is required to satisfy a new request for a segment. The memory manager looks for a segment to overlay if no free area of the required size is found by searching the list of free areas. If there are any segments on the overlay selection list these will be overlayed one at a time until a space has been created that is large enough to satisfy the request. Overlaying a segment involves ensuring that the segment has been copied back to virtual memory if necessary, releasing the space it occupied in main memory, and coallescing the free space created with any adjacent free spaces that might exist. Special dummy links are provided at bank boundaries to appear busy to the memory manager, so that free areas will never be able to span banks; it is this simple mechanism that ensures that any segment will always be wholly contained within a bank. If another free area is found to be separated from the newly created one by one small movable segment then that segment will be physically moved in main memory to allow for combination of the two free areas. If the overlay selection list becomes exhausted before a large enough free space is found the memory manager must turn elsewhere for help in predicting which segments in main memory will not be used in the near future. At this point it is known that all segments in memory are actually required by some process for it to run well. The memory manager now has no choice but to remove one of the processes from main memory temporarily. A communication mechanism has been set up with the dispatcher to assist in predicting which process is least likely to run in the near future.

When the dispatcher puts a process to sleep it decides, knowing the reason for suspension of the process, whether that process is likely to wait for a long period before reactivation. If a long wait is likely, the dispatcher links that process to the end of a list called the discard list. The memory manager knows that a process on this list is taking up main memory but is unlikely to need it soon. Processes are discarded by selecting them from this list one at a time and overlaying each segment in their working sets starting with the least recently used. This procedure is carried out until enough space has been released to satisfy the request on which the memory manager is working. If the discard list is exhausted before enough space is found the memory manager can, as a last resort, scan the dispatcher's ready list and discard processes starting with the one having the lowest priority. In this way working sets of the processes highest on the ready list will remain in memory; these are precisely the processes the dispatcher will schedule next, and thus the memory manager is actually using some foreknowledge of the near future to assist in its predictions. Of course this knowledge is not perfect in an interrupt-driven system like the HP 3000, where processes can be moved onto the ready list at any time. The best that can possibly be achieved in predicting the future is to maximize the probability of being correct, using information from the recent past.[5] Performance measurements of the system under a typical load show that the strategy used is indeed efficient. The cost of having memory managed completely by the operating system varies, depending on the amount of real memory available, from less than 5% of the CPU time on the larger configuration to about 12% of the CPU time on the smallest system.

Other Changes

Since the availability of Schottky TTL had improved and the CPU had to be changed anyway the microprocessor was redesigned for increased speed. This was achieved by speeding up access to memory operands, and by modifying the pipeline[1,6] to minimize time spent waiting for the pipeline to empty when a microcode jump occurs. This change did not affect the normal (unbranched) operation of the pipe and so the microprocessor still takes advantage of the

inherent parallelism of the pipe. In addition, a decimal arithmetic circuit was added to the microprocessor to assist in the execution of the new decimal instructions that were added. The overall effect of these changes and the faster memory is to increase the average number of instructions executed per second by 50%.

As a result of extensive performance measurements on the previous system a number of new privileged instructions were added to assist the operating system in improving its efficiency. This includes a set of instructions for moving data between data segments without having to alter any of the registers. There are also instructions for manipulating system tables while eliminating the multiple memory references previously performed explicitly by the software.

Another class of new instructions that has had a major effect on the overall performance of the Series II consists of process handling instructions added to facilitate the control of process switching. These include instructions to disable, enable, and invoke the dispatcher. One instruction, IXIT, now performs all the functions previously performed by the most common path through the dispatcher. After an interrupt has been processed it is necessary to dispatch a process. Previously the dispatcher had to be invoked to do this. On the Series II the interrupt handler need only execute the IXIT instruction to redispatch the interrupted process. If the interrupt was important enough (e.g., a higher-priority process becomes active) a DISP instruction is issued prior to the IXIT; this invokes the dispatcher to decide which process to run next. These instructions, combined with a redesign of the dispatcher and its scheduling mechanism, have resulted in a dramatic reduction in the time required to switch processes. A full dispatch, consisting of terminating one process, invoking the dispatcher, updating the CPU time used for that process, determining which process to run next, setting up the environment for that process, and finally launching it takes less than a millisecond.

Further reductions in operating system overhead have been achieved by a redesign of the software input/output system. The changes minimize the number of process switches required to perform an I/O operation in addition to optimizing the code itself. The result is an extremely efficient I/O structure requiring little CPU processing for I/O transfers. The most apparent effect of this improvement is that it is now possible to run spoolers at higher priority than users. This means that the Series II can continuously spool output to the fastest line printer with no noticeable impact on the performance of the system. In addition, character interrupts from asynchronous terminals can now be processed three times faster than before, which increases the number of terminals that can be supported by the system.

Acknowledgments

A product of this complexity can be a success only with the contributions of many good people. Contributing to the hardware design of the HP 3000 Series II were Rick Amerson, Bill Berte, Wally Chan, Don Jenkins, Elio Toschi, Tak Watanabe, and Steve Wierenga. John Sell and Ron Kolb were mainly responsible for the microcode. Joel Bartlett, Larry Bierenbaum, Russ Blake, Tom Blease, Hank Davenport, John Dieckman, Tom Ellestad, Pete Gragiano, Mike Green, Bob Heideman, Alan Hewer, Ron Hoyt, Jack MacDonald, Dennis McEvoy, Chuck Robinson, and Karel Siler worked on various aspects of the operating system.

References

1. Hewlett-Packard Journal, January 1973.
2. W.W. Chu and H. Oderbeck, "The Page Fault Frequency Algorithm," Proc. FJCC 1972, pp 597-609.
3. P.J. Denning, "Virtual Memory," Computing Surveys, Vol. 2, No. 3, September 1970, pp 153-188.
4. G.S. Graham and P.J. Denning, "Multiprogramming and Program Behavior," Sigmetrics Symposium on Measurement and Evaluation, 1974, pp 1-8.
5. L.A. Belady, "A Study of Replacement Algorithms for a Virtual Storage Computer," IBM Sys. Journal 5,2, 1966, pp. 78-101.
6. J.V. Sell, "Microprogramming in an Integrated Hardware/Software System," Computer Design, 1975, pp. 77-83.

An All-Semiconductor Memory with Fault Detection, Correction, and Logging

by Elio A. Toschi and Tak Watanabe

DESIGN OBJECTIVES FOR THE HP 3000 Series II memory system included high speed, low cost, small size, high reliability, and low maintenance. The speed, size, and cost goals were met by using a 4096-bit N-channel metal-oxide-semiconductor random access memory, commonly called a 4K MOS RAM, as the fundamental building block. Fault detection, correction, and logging were added to further improve reliability and reduce maintenance requirements.

Memory Organization

Memory subsystems in HP 3000 Series II Computer Systems are independently functioning modules. There are one or two memory modules per computer system. Each memory module consists of three types of printed circuit boards:

- one memory control and logging board (MCL)
- one fault correction array (FCA)
- one to four semiconductor memory arrays (SMA).

The MCL contains bus interface logic, data and address registers, timing and refresh logic, and fault correction and logging logic. The FCA contains Hamming code generators, address and data drivers, and four 32K-word×4-bit MOS RAM arrays. The four arrays on the FCA supply most of the additional bits per word necessary for fault correction. The SMA contains a 32K-word×17-bit MOS RAM array, and address and data drivers. The FCA and the SMA together form the 21-bit memory words. Each word consists of 16 data bits and five check bits.

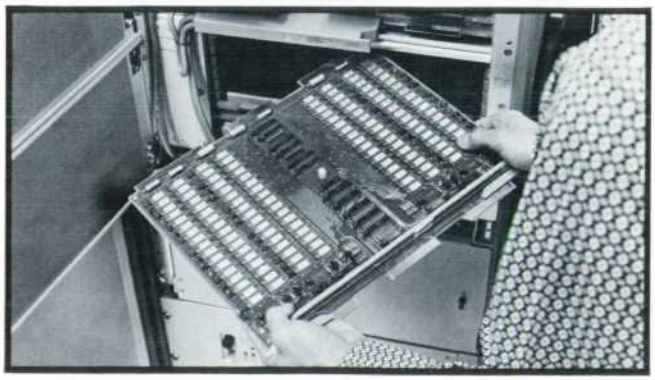

Fig. 1. *136 18-pin 4K RAMs on each HP 3000 Series II memory board provide 32,768 17-bit words of semiconductor memory capacity, four times the capacity of the same-size board with core memory. Series II memory is 50% faster and less than one-third as costly as the older core memory.*

The memory module is expandable in 32K-word increments. There can be up to four SMA boards per module and up to eight SMA boards per computer system. Along with the two memory modules, the memory system contains a fault logging interface (FLI) board that interfaces the logging logic on the two MCL boards to the I/O system.

Why Semiconductor Memory?

Several aspects of the 4K MOS RAM make it attractive for main memory. Among these are low cost, high speed, high density, and good long-term reliability.

Cost. Core has been used in computers for some 20 years and breakthroughs in cost seem unlikely. The ratio of the cost/bit of core to that of semiconductor RAM is approximately 3 to 1 today.[1] Since 4K RAM manufacturers are still on the steep part of the learning curve this ratio should continue to increase. Also, many of the necessary overhead circuits (drivers, decoders, timing) are incorporated within the 4K RAM, so fewer external overhead circuits are needed. Reduced external circuitry means reduced manufacturing costs and ultimately cost savings to the user. Series II memory costs less than one-third of the core memory previously used in the HP 3000.

Performance. In the Series II, an overall 30% improvement was achieved in memory system access and cycle time over the previous HP 3000 core memory (access 300 ns vs. 525 ns, cycle 700 ns vs. 1050 ns). These speed improvements include the overhead time necessary for fault detection and correction.

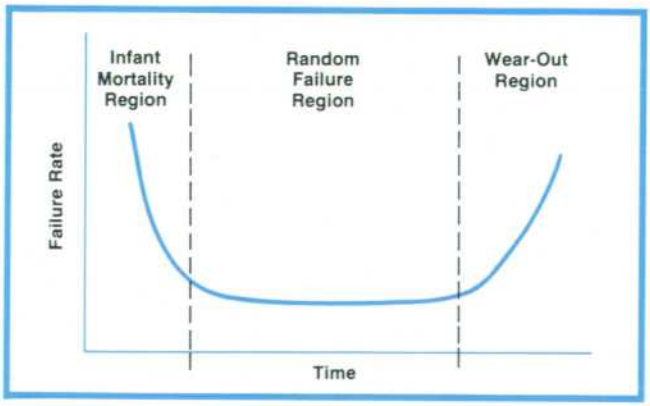

Fig. 2. *4K RAMs follow the well-known reliability life curve. Error correction reduces the effects of memory failures on system performance in all regions of the curve.*

High density. Using the same printed circuit board format as the core memory (see Fig. 1), the board word capacity was increased by a factor of four using 4K RAMs. The result is a substantially higher-capacity memory in basically the same volume as the older core memory.

Reliability. The goal was to make the new memory subsystem significantly more reliable than the older HP 3000 core memory. This was achieved in several ways. First, by taking advantage of the fact that much of the overhead logic is incorporated in the 4K RAM and by using MSI (medium scale integration) logic wherever possible, the parts count was greatly reduced. The 256K-word semiconductor memory including error correction requires approximately 25% fewer components than the 64K-word HP 3000 core memory. Second, error correction logic was added to maximize and stabilize 4K RAM reliability.

4K RAMs follow the well known reliability life curve (see Fig. 2). In the early stages of their life there is a high-failure-rate region known as infant mortality. Accelerated aging (at high temperature) and stringent testing are used to weed out most of the failures and weak 4K RAMs in this region before shipment of the computer. But it still takes time to reach the random failure region where failure rates are very low and stable. Error correction minimizes system crashes caused by memory failures in the infant mortality region. Once past the infant mortality region (1000 to 1500 hours), memory with error correction should become extremely stable, since the random failure region is estimated to last from tens to hundreds of years.[2]

Volatility. One undesirable aspect of semiconductor memory is volatility, that is, unless power is continuously applied to the 4K RAMs, stored data is lost. In the Series II, critical voltages to the 4K RAMs are backed up with a battery. When ac power is lost the sensors in the power supplies force the computer into a power-fail routine. Upon completion of the power-fail routine the memory goes into a protected mode and all critical voltages are switched to battery power. When ac power is restored the computer automatically restarts and the battery is switched to a rapid recharge mode, returning to 90% of full capacity within 1½ hours.

How Error Correction Works

Error correction requires that redundant information be added to the data word. The minimum number of additional bits required for single-error correction is governed by the equation:

$$2^K \geq m+K+1$$

where m = number of data bits, and K = number of Hamming parity bits or check bits.[3] Solving the equation for K where m = 16, a minimum of five check bits are needed for single error correction when there are 16 data bits.

In the Series II, when a word is written into main memory, five additional bits called the check bit field are added to the word. These five check bits are derived from a parity generator called a Hamming generator, which constructs the check bit field from selected fields of the word. Check bit field generation is shown in Fig. 3.

Check bit field generation takes place in three different boards: MCL, FCA, and the accessed SMA board. The data path is shown in Fig. 4. As the data is written into the RAMs of the SMA board it is also presented to the FCA board where the Hamming generator calculates the check bit field. Four of the check bits are stored on the FCA board. The remaining bit is stored on the SMA board.

When a word is read from memory the data and the check bit field are used to compute an error code. If the error code is 0's then there are no detectable errors in the 21-bit word. If the five bits of error code are other than zeros an error exists, and the error code uniquely pinpoints the bit in error (see Fig. 5).

The error code is stored and decoded on the MCL

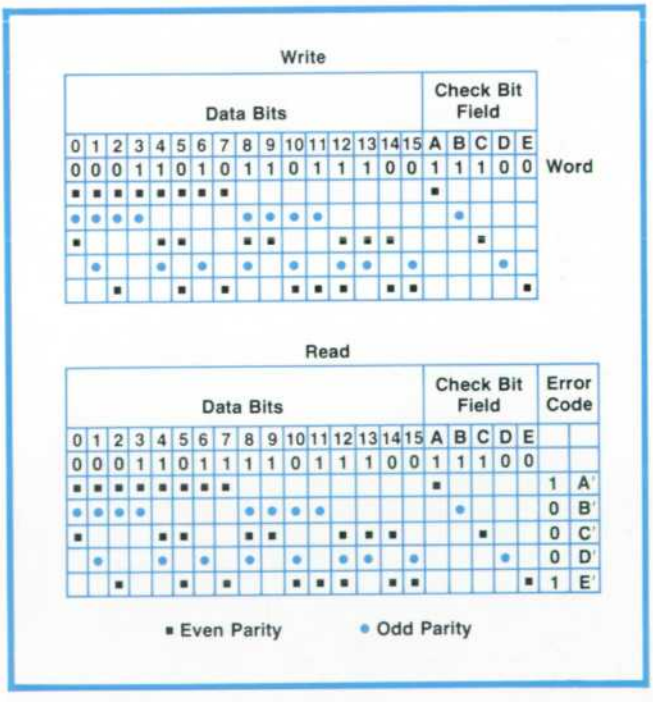

Fig. 3. *For error correction, five redundant bits, the check bit field, are added to each 16-bit data word when it is written into memory. The five check bits are generated as even or odd parity bits for various data fields. For example, in the word shown, there are three data ones in bits 0-7 and even parity (an even number of ones) is required, so check bit A is a one. Check bit B is a one because there are four ones in bits 0-3 and 8-11 and odd parity is required. When the word is read from memory an error code is calculated. If it is other than all zeros an error has occurred. Here data bit 7 has been read incorrectly.*

board. If the code points to a data bit, that data bit is complemented by an EXCLUSIVE OR gate, thus changing its sense.

An example is shown in Fig. 3 of how error correction works. For the data word shown, the check bit field is 11100. The check bit field is stored along with the data at the same address. As that location is read, the data and check bit field are used to compute an error code of 10001. In this case bit 7 was stored as a 0, but read out as a 1. By decoding 10001 using Fig. 5 we see that data bit 7 is in error and should be complemented.

Detecting and correcting errors does not increase the access time or the cycle time of the memory system.

Error Logging

When an error is detected during a read cycle, a one is written in the logging RAM at the address derived from the error code concatenated with the five most significant bits of the main memory address. The 1024 locations of the logging RAM have unique physical significance. All single-bit failures can be traced down to a board, a 4K RAM row, and a bit. All other failures can be traced to a board or boards and a 4K RAM row. Fig. 5 is a map of the logging RAM showing the result of detecting the error described above with the example carried a step farther to include address information and bit location.

The logging RAM consists of one 1024×1-bit chip on the MCL board. It provides enough locations for all the memory associated with one MCL (128K words). Error logging is accomplished simultaneously with the normal memory cycle operation and requires no additional time.

Under MPE (the Multiprogramming Executive operating system), the HP 3000 Series II I/O system may interrogate the lower logging RAM (for the lower 128K words of memory) or the upper logging RAM (for the upper 128K words of memory). The logged information is copied into another 1024 × 1 RAM used for temporary storage. Any 1's in this RAM act as error flags. If a flag is detected the RAM address (fault data) is read by MPE. The fault data is tabulated and is printed out as the Error Correcting Memory Log

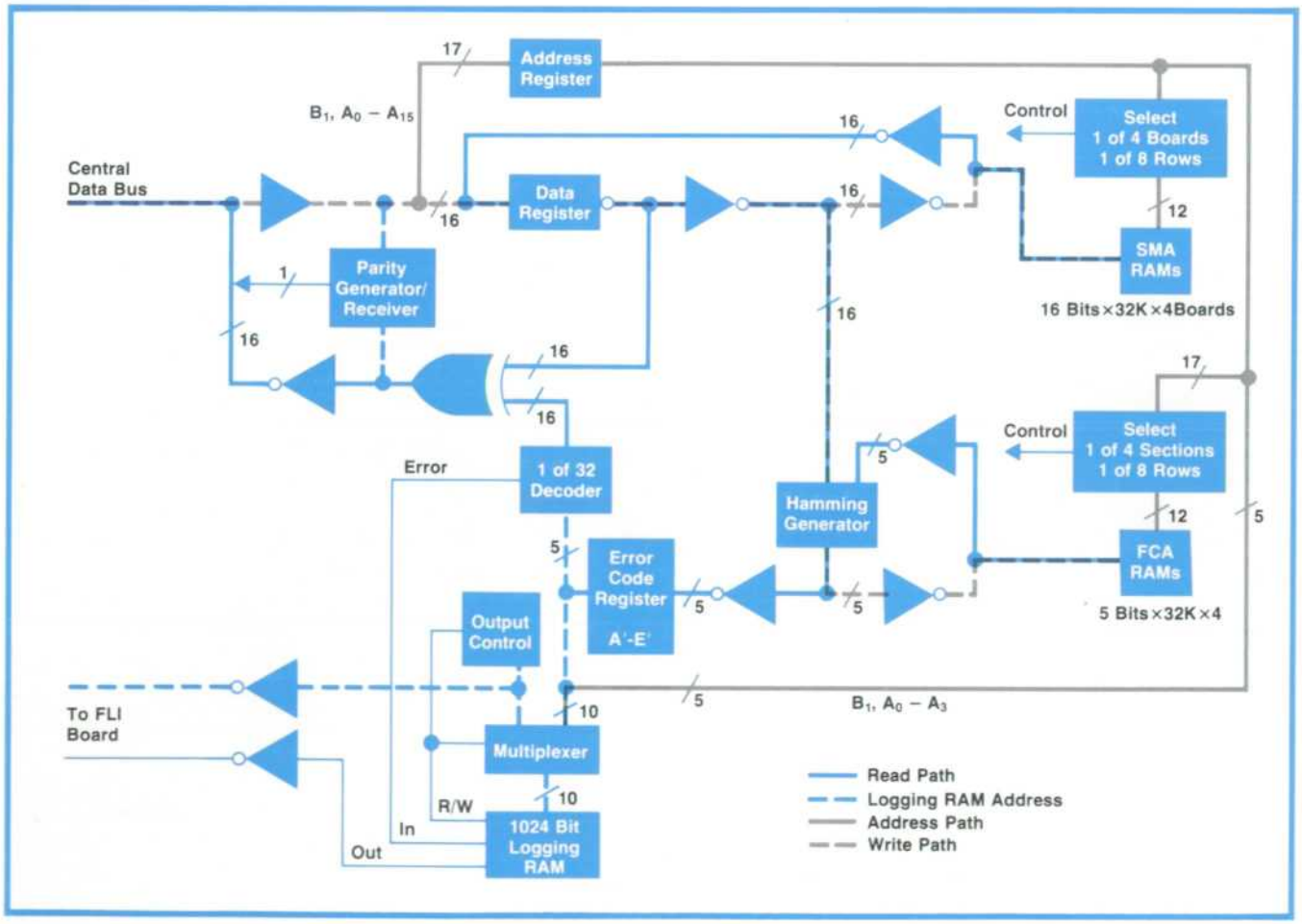

Fig. 4. *Check bit field generation takes place in three memory-module boards: MCL, FCA, and the accessed SMA board. The Hamming generator on the FCA board does the calculations. Four check bits are stored on the FCA board and one on the SMA.*

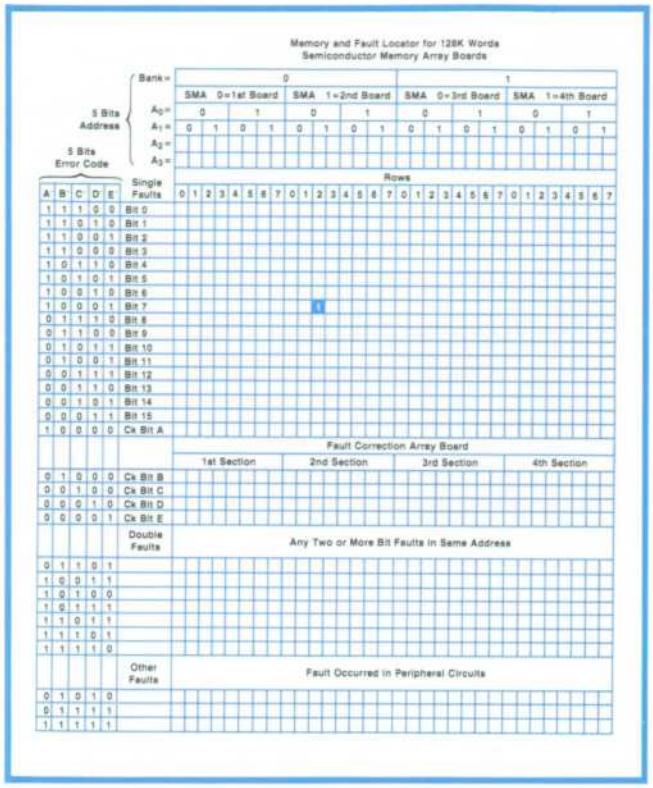

Fig. 5. *The error code pinpoints the bit in error for all single-bit errors, as shown at left. All single-bit errors are corrected automatically and logged in a 1024-bit RAM, as shown here for the example of Fig. 3.*

Analysis (Fig. 6).

The fault-logging RAMs are interrogated periodically (about once per hour). Very little CPU time is used.

Types of RAM Failures

The effectiveness of the error correction logic depends on the failure modes of the 4K MOS RAM. Error correction is most effective if all errors are single-bit failures with no address sensitivity (that is, minimum multiple-bit failures). Data gathered by RAM manufacturers and by Hewlett-Packard indicate that a large majority, approximately 75% to 80%, of 4K RAM failures are single-bit.[4]

There are two types of single-bit failures. The first is a hard failure, in which a memory cell is "dead". Although this type of failure is a potential problem for error correction because it fails every time it is addressed and increases the probability of a double-bit failure, it is easily located and removed. The other type of single-bit failure, the soft failure, is far more difficult to locate. These failures tend to be nonrepeatable or occur very infrequently. Soft failures are caused by chip susceptibility to noise, voltage and temperature variations, data patterns, timing, and read/write sequencing. For example, if a 4K RAM is sensitive to a particular data pattern, it will fail only when that pattern is present in the RAM. That particular pattern may be very difficult to reproduce. This is why one of the most difficult tasks in semiconductor memory design is devising effective diagnostics.

With error detection and logging, diagnostics become inherent in the memory design. When a failure occurs during normal operation it is automatically logged. This does not mean that memory diagnostics are no longer necessary, but that single soft failures are no longer a critical failure mode. Also, soft single-bit failures do not reduce the effectiveness of error correction to the same degree that hard failures do.

Multiple-cell failures within a 4K RAM are potentially most hazardous to the effectiveness of error correction. The least desirable multiple-cell failure is a totally "dead" 4K RAM. Fortunately, this type of failure occurs very infrequently and is easily detected and repaired.

Reliability Improvement with Error Correction

The reliability of a semiconductor memory subsystem is typically a direct function of the number of 4K RAMs in that subsystem. Hence as the memory size increases, memory reliability can become the limiting factor on the overall computer system reliability. Also, the reliability of 4K RAMs varies from maker to maker and even between lots from a given manufacturer because of process and circuit changes. Ideally, one would like to design a memory subsystem that has a reliability independent of memory size and 4K RAM reliability. Error correction does much to achieve this goal.

The following definitions are necessary for a quantitative discussion of reliability.

- Failure rate (λ) is the average percentage of all devices that can be expected to fail per unit of time. Failure rates are usually expressed in percent per thousand hours. Semiconductor devices exhibit changing failure rates with time, so a time frame is usually specified along with a failure rate.
- Mean time between failures (MTBF) is the reciprocal of failure rate.
- Mean time to repair (MTTR) is the average time required to fix a system once a failure is detected.

The effective failure rate of an error-corrected 4K RAM array is the rate of multiple-bit failures, since all single-bit failures are corrected. A multiple-bit failure might consist of two 4K RAMs failing at the same subsystem address. Totally dead 4K RAMs contribute most to increasing the multiple-bit failure rate while single-bit failures contribute least.

Separating the failure rates for the 4K RAMs and for the peripheral logic we have for the non-error-corrected memory:

$$\lambda_{T4KRAM} + \lambda_{TLOGIC} = \lambda_{NECSYS}$$

where λ_{T4KRAM} = failure rate of 4K RAMs in subsystem

λ_{TLOGIC} = failure rate of logic ICs in subsystem

λ_{NECSYS} = failure rate of total subsystem without error correction.

For the error-corrected memory,

$$\lambda_{T4KRAM} \| \lambda_{EC4KRAM} + \lambda_{ECLOGIC} + \lambda_{TLOGIC} = \lambda_{ECSYS}$$

where $\lambda_{EC4KRAM}$ = failure rate of error correction path 4K RAMs

$\lambda_{ECLOGIC}$ = failure rate of error correction logic ICs

λ_{ECSYS} = failure rate of total subsystem with error correction.

The error correction RAMs are redundant and can be thought of as forming a parallel path with the data RAMs. The 4K RAM has 4096 addresses. A single-bit failure is considered to be a failure in any one of those addresses. The multiple-bit failure rate is dependent on the frequency with which one repairs single-bit failures (MTTR) and the number of cells that fail in a 4K RAM when a failure occurs. Using statistical data gathered by 4K RAM manufacturers and by HP the multiple-bit failure rate in percent per 1000 hours is plotted against MTTR (single-bit) for a 128K-word 4K RAM array in Fig. 7. The parameter in these curves is the basic failure rate, λ_{4KRAM}, of a single 4K RAM. For example, assuming that detected single-bit failures can be repaired within one month, or 720 hours, MTTR (single-bit) = 720. If $\lambda_{4KRAM} = 0.1\%$ per 1000 hours, then from Fig. 7, $\lambda_{T4KRAM} \| \lambda_{EC4KRAM} = \lambda_{ECTRAM}$ = 0.17%/1000 hours (or MTBF = 67 years) for a 128K-word array. This figure is much smaller than $\lambda_{TLOGIC} + \lambda_{ECLOGIC}$. That is:

$$\lambda_{ECLOGIC} + \lambda_{TLOGIC} \approx \lambda_{ECSYS}.$$

Thus it can be seen that error correcting tends to stabilize the memory subsystem and make it relatively independent of the 4K RAM failure rate.

The improvement of memory subsystem reliability with error correction is $\lambda_{NECSYS} \div \lambda_{ECSYS}$. By knowing the λ for each part and the part count we can tabulate an improvement factor.

	Memory Size		
	64K	128K	256K
λ_{4KRAM}	Improvement		
0.05%/khr	3	4	4
0.2%/khr	9.8	15	15

Data gathered at HP and by IC manufacturers indicates that λ_{4KRAM} after a few thousand hours of operation is between 0.05% and 0.2% per 1000 hours, making the overall memory subsystem between 3 and 15 times more reliable than a system without error correction.

From Fig. 7, MTTR directly affects the percent failure per 1000 hours. This is a parameter over which we can exercise complete control. During the first few thousand hours of operation the failure rate for the 4K RAMs is high, but it is possible to compensate for the higher rate by decreasing MTTR. Since the system keeps track of all errors, error logging can work as a feedback mechanism whereby fewer errors will require less maintenance or more errors require more frequent maintenance. The result of constantly monitoring the system can be used to calculate the MTTR required to achieve the desired low probability of ever getting a multiple-bit failure. For example, using a mature failure rate of 0.05%/1000 hours for the 4K RAM, as might be expected after a few thousand hours of operation, and an MTTR of one month, Fig. 7 indicates a multiple-bit failure rate of 0.04%/1000 hours or an MTBF for multiple-bit failures of 285 years for the RAMs of the memory subsystem. The rest of the memory subsystem will have an MTBF determined by its IC logic.

Low Maintenance

Since fault correction essentially prevents a computer from failing even though a 4K RAM has failed, memory maintenance can be postponed and performed at a normally scheduled time or at a time that is more convenient for the user. When memory maintenance and repair is necessary the customer engineer, who services the computer, will find his task much easier.

Since every fault that is detected is logged and tabulated by the operating system, very accurate data on the failing devices is maintained. The customer engineer can use this data in several ways. First, without running elaborate diagnostics, he immediately knows all the detected failures that occurred in the memory subsystem under the user's operating environment. Second, he knows which 4K RAMs are potential troublemakers, since logging tells him how many times each failure was logged. The failure count is important because not every failure requires replacement of a 4K RAM. For example, if a 4K RAM fails only once in a month it probably would not have

```
ERROR CORRECTING MEMORY LOG ANALYSIS

LOGGING STARTED     - DATE:  7/ 2/76   TIME:  8:53
FIRST ERROR LOGGED  - DATE:  7/ 2/76   TIME:  8:53
LAST ERROR LOGGED   - DATE:  7/ 2/76   TIME:  9:04
LAST LOG UPDATE     - DATE:  7/ 2/76   TIME:  9:04

-------------------------------------------------------------------
I       ADDRESS         I              ERROR TYPE              I ERROR I
-------------------------------------------------------------------
I MEMORY I  LOC  I ROW I  TYPE   BIT   BOARD   CHIP  I COUNT I
-------------------------------------------------------------------
I RANK 0 I  U32K I   2 I  DATA    7     SMA    U710  I    23 I
I        I       I   4 I  DATA    4     SMA    U47   I     1 I
-------------------------------------------------------------------
```

Fig. 6. *3000 Series II Systems log all corrected errors. The printed error log helps the customer engineer weed out 4K RAMs that fail too often.*

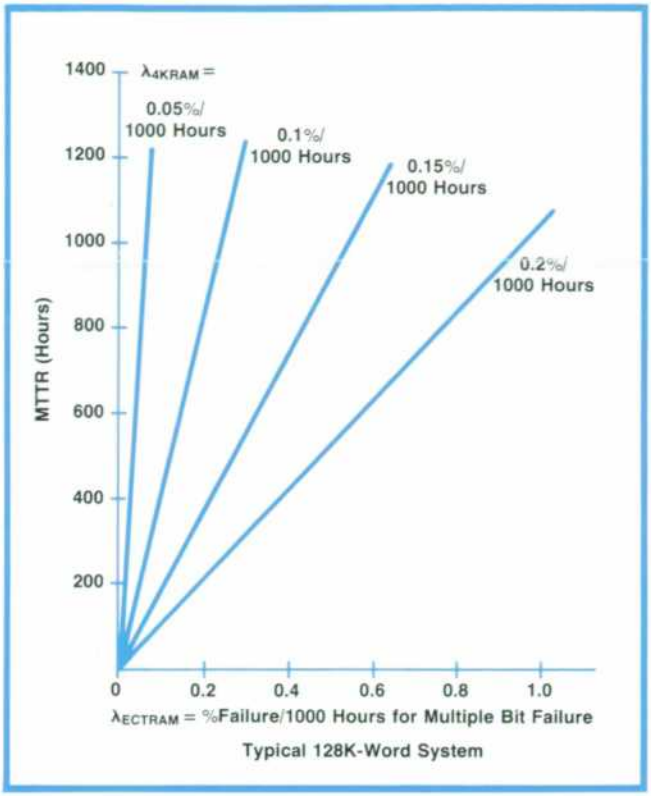

Fig. 7. *The expected multiple-bit failure rate depends on the failure rate of individual 4K RAMs and the mean time to repair single-bit failures (MTTR).*

to be replaced, but if a unit fails, say, 10 times in a month it probably should be replaced because it will reduce the effectiveness of fault correction. Third, the log tells the customer engineer exactly which 4K RAM to replace (see Fig. 6). This eliminates human errors in interpreting the data. Fourth, after the customer engineer repairs the memory by replacing any defective 4K RAMs, he leaves the user with a more reliable system, that is, the reliability of the memory improves because the weak RAMs are gradually being weeded out of the system. Because memory boards are repaired on site, the user retains a computer with known memory reliability instead of a computer with an exchange board of unknown age and unknown reliability.

Acknowledgments

We would like to thank all of the people who contributed to the success of this project. The design team included Slava Mach, Bob Heideman, Ron Kolb, Barney Greene, Don Jenkins, Ron Hoyt, and John Sell. Printed circuit layout was done by John Bruno, Pat Mansfield, Barbara Martin, Butch Zottoli, Lois Ghan, and Marianne Miller. Production engineering was done by Curt Gee, Paul Chang, and Dan Mathias.

Special acknowledgments go to Bill Berte for his contributions as a member of the design team and to Rick Amerson for his technical advice.

References

1. R.J. Frankenberg, "Designer's Guide to Semiconductor Memories—Part 3," EDN, 1975.
2. W. Pascoe, "2107A/2107B N-Channel Silicon Gate MOS 4K RAM," Reliability Report RR-7, September 1975.
3. W.W. Peterson, "Error Correcting Codes," M.I.T. Press, 1961.
4. T.L. Palfi, "MOS Memory System Reliability," IEEE 1975 Semiconductor Test Symposium.

Leonard E. Shar

By the time this is printed, Len Shar expects to be in Australia on the first leg of a trip around the world. Before going on leave, he was project manager for HP 3000 operating systems. A native of Johannesburg, South Africa, Len received his BSc degree in electrical engineering from the University of the Witwatersrand in 1968, and his MS and PhD degrees in computer science from Stanford University in 1970 and 1972. He's been with HP since 1972, is a member of IEEE, and has taught logic design at Stanford. For relaxation, he likes hiking, bicycling, reading, and music.

Tak Watanabe

With HP since 1970, Tak Watanabe was a principal designer of the original core memory and the new semiconductor memory for the HP 3000. Tak attended the University of California at Berkeley, graduating in 1966 with a BSEE degree. He designed missile systems for three years, and in 1970 he received his MSEE degree from San Jose State University. Born in Lone Pine, California, he's single and now lives in Sunnyvale, California. His hobbies are photography and designing high-fidelity audio equipment.

Elio A. Toschi

Elio Toschi, one of the principal designers of the HP 3000 error correcting memory, has been designing computer hardware for HP since 1965. After receiving his BS degree in electrical engineering from San Jose State University in 1957, and before coming to HP, he designed avionics equipment for eight years. He's a member of IEEE. Elio was born in San Jose, California and still lives there. He's married and has two children, and he loves skiing of any variety, snow or water.

HP 3000 Series II Performance Measurement

by Clifford A. Jager

The HP 3000 Series II hardware, firmware and software were designed to provide increased performance and capabilities over the HP 3000CX product line. Measurements of these new performance levels have been conducted to confirm Series II design objectives and to better define Series II performance. Of course, because of the great number of operational variables in the use of a general-purpose multiprogramming computer system such as the HP 3000 Series II, the full extent of its performance capabilities cannot be defined precisely.

Early Measurements

Preliminary measurements made a year ago confirmed the objective of increasing the general instruction set speed by about one-third. The average instruction time during an SPL (HP 3000 System Programming Language) compilation went from 4.08 microseconds to 2.57 microseconds. This meant that the Series II had an increase in throughput capacity of 50% without considering the main performance enhancing factor, that is, up to four times the amount of main memory.*

As Series II hardware and software became increasingly reliable, measurements were made involving many users and various memory sizes. In single-subsystem measurements using the COBOL compiler and BASIC interpreter improvements of as much as 10 to 1 were seen in throughput and response time.

TEPE

The Teleprocessing Event Performance Evaluator (TEPE) System[1] was used to conduct these early measurements and the others discussed below. It was developed at HP's Data Systems Division and consists of a program that runs on an HP 2100 Computer-based system and simulates up to 32 terminal users. TEPE and the driven system, in this case a 3000 Series II, are hard-wired together and each terminal user's actions are prescribed by a script. All messages that pass between TEPE and the driven system are time-stamped and recorded on magnetic tape that is later processed by data reduction programs.

TEPE can simulate terminals of various speeds by sending delay messages to the driven system to simulate user typing and impose user think time between transactions. In the measurements described below all users, both batch jobs and interactive sessions, were administered by TEPE. All were run using simulated 2400-baud terminals. Input to the Series II was delayed 0.3 seconds per character to approximate typing. Inter-event user think time for jobs was a constant 1 second, while sessions used a random think time exponentially distributed from one to 94 seconds with a mean of 22 seconds.[2] All job output that would normally go to a line printer was directed there via a FILE command.

New Scripts

The COBOL and BASIC scripts used in the early measurements were useful indicators but would probably not be considered typical by a potential user. Therefore several new scripts were devised with the aim that they address a general application area, be reasonable, and represent work loads that could run on all standard Series II configurations so as to differentiate their capabilities.[3] Since these scripts were designed to run on all Series II configurations, they do not represent the maximum workload for any Series II configuration. No script was contrived or tuned and no program was optimized.

Three scripts were constructed, one for each of the following user environments:

User Environment	Total No. Users	No. Batch Jobs	No. Interactive Sessions
Timesharing	15	2	13
Scientific	10	4	6
Commercial	7	4	3

*If an arbitrary amount of work W is completed in 1/3 less time T and the original throughput is defined as the rate of doing work, W/T, then the new throughput is W/T (1-1/3) which is 1½ W/T, or a 50% increase.

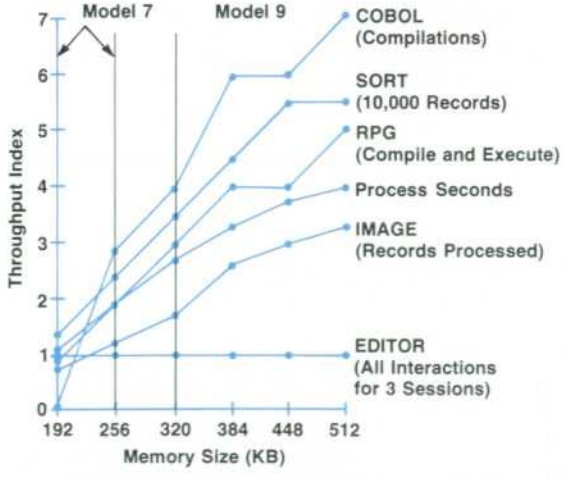

Fig. 1. *Relative throughput of HP 3000 Series II models 7 and 9 using several measures of throughput. The base is a model 5 system with 128K-byte memory.*

User		Event	
No.	Type	Unit	Completions
12	Standard BASIC Mix	> RUN	127
		? Data	636
		> Add Statement	535
		> LIST 27 Statements	133
		> GET	68
		> SAVE!	57
		> RENAME	70
		: BASIC	68
		> PURGE	56
		> EXIT	56
1	Compiled BASIC Program	: RUN	7
1	Interpreted BASIC Program	> RUN	5
1	List 23-Chain BASIC Program	Listings	55

Fig. 2. *Absolute throughput in one hour on a 512K-byte model 9 for the timesharing script.*

The scripts had several features in common. Each was a mixture of jobs and sessions, used several Series II subsystems, and generated spooled output to a line printer. Each was used to measure response time and relative and absolute throughput. In these scripts, no user's activity ever ceases; as a cycle completes it starts again.

The configurations tested were the standard models 5, 7 and 9. No optional configurations or additional equipment were used except that all optional memory sizes were measured. All models had extended instruction set (EIS) firmware. The model 5 had one HP 7905A disc drive and the models 7 and 9 had two HP 2888A disc drives. Memory sizes tested were 128K, 192K, and 256K bytes on the model 5, 192K and 256K bytes on the model 7, and 320K, 384K, 448K, and 512K bytes on the model 9.

Definition of Terms

Response time is the time from the initiation of a request until the system is ready to accept the next request. In other words, it is the time from the carriage return terminating one request until the prompt beginning the next. Response time does not include user think time or delays to simulate typing. It does, however, include the time for all responses to a given request.

Throughput is the rate of doing work. The units of work and time may be somewhat arbitrary. In the early COBOL measurements, the unit of work was a compilation and the unit of time was an hour. Compilations, in this case, were a useful measure of work since they occurred frequently with respect to the duration of the measurement and they were homogenous across all users. When events become infrequent or dissimilar they are not quite so useful for measuring the throughput of a system. Therefore, a common unit of work, the process second, will also be used to describe throughput. Process time is that time when any process operates on behalf of an individual user whether he is doing computation or input/output operations. It does not include system overhead for administering multiprogramming or pause time when no user is able to run. The sum of process seconds allocated to all users in an elapsed hour will be used to describe throughput.

User		Event	
No.	Type	Unit	Completions
3	Standard Basic Mix		
		>RUN	32
		?Data	164
		>Add Statement	133
		>LIST 27 Statements	33
		>GET	18
		>SAVE!	15
		>RENAME	18
		:BASIC	18
		>PURGE	15
		>EXIT	15
3	EDITOR	All	398
1	Compiled BASIC Program	:RUN	5
1	Interpreted BASIC Program	>RUN	4
1	BASIC Compile and Run	:BASICGO	0
1	FORTRAN Compile and Run	:FORTGO	22

Fig. 3. *Absolute throughput in one hour on a 512K-byte model 9 for the scientific script.*

User		Event	
No.	Type	Unit	Completions
1	COBOL Compile	:COBOL	7
1	SORT	10K Records	11
3	EDITOR	All	338
1	RPG Compile and Run	:RPGGO	5
1	IMAGE	Input Records	826

Fig. 4. *Absolute throughput in one hour on a 512K-byte model 9 for the commercial script.*

Fig. 1 is an attempt to describe relative throughput for the commercial script using several dissimilar events. Results are relative to the number of events completed in one hour on a model 5 with minimum memory.

The events completed by the three editor sessions remain constant since they have high priority, require little processor time, and are essentially think-time bound. It appears that COBOL and RPG made no improvement between 384K and 448K bytes, but the fractional part of the next event completed is unknown. In general the throughput for COBOL, SORT, and RPG is probably overstated because of even greater error in fractional parts of the base configuration. Process time (i.e., process seconds) then, is one convenient way to combine these dissimilar activities into a single representation with very little error. It solves the problem of the partially completed event and the summing of unlike events.

Results

Figs. 2, 3, and 4 show throughput for the three scripts in absolute terms for the model 9 with 512K-byte memory. Figs. 5, 6, and 7 show relative throughput and response time for all three models. The precision or repeatability of the various experiments was checked at several points. Throughputs measured by process seconds agreed within 1% while mean response times agreed within 5%.

The results for the three scripts are quite similar, showing that relative throughput ranges from 1 to 3½ or 4 with an accompanying improvement (decline) in response time as the memory size

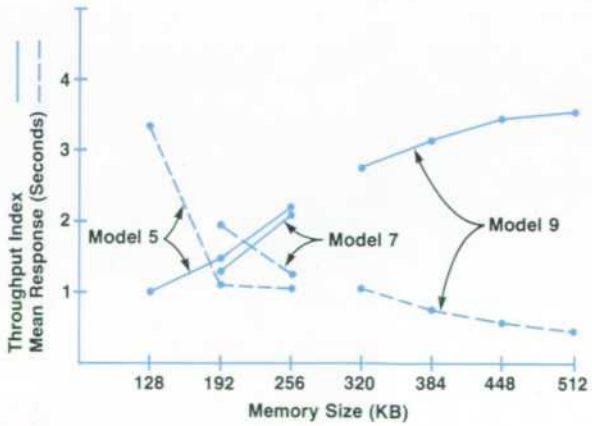

Fig. 5. *Relative throughput and response time for models 5, 7, and 9 for the timesharing script. Response time is based upon statement entry and modification for 12 BASIC sessions in a standard mix of operations.*

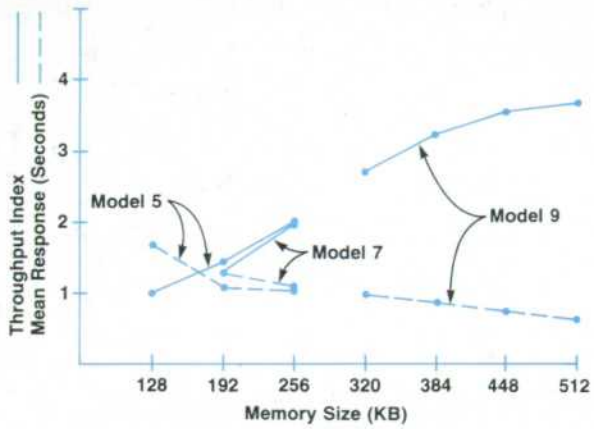

Fig. 6. *Relative throughput and response time for models 5, 7, and 9 for the scientific script. Response time is based upon all events in three EDITOR sessions.*

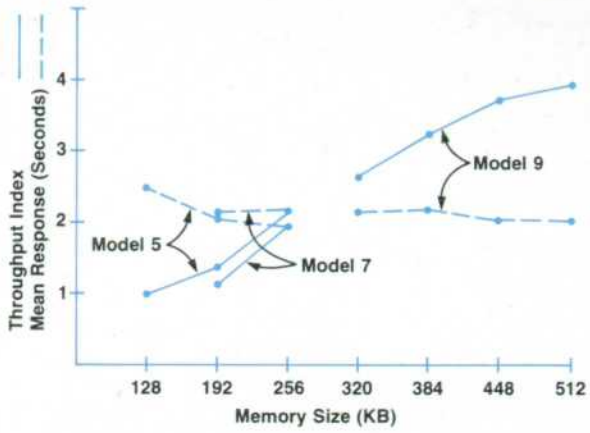

Fig. 7. *Relative throughput and response time for models 5, 7, and 9 for the commercial script. Response time is based upon all events in three EDITOR sessions.*

increases. An exception is that throughput and response time favor the model 5 over the model 7 for equivalent memory sizes. This is because of their disc configurations. The model 7, which is designed for commercial users of moderate size, has six times the disc storage space of the model 5, but slower disc drives, access speed having been sacrificed for capacity.

Among the conclusions that may be drawn from these experiments is that performance of the Series II begins approximately where that of the 3000CX ends and exceeds it by as much as an order of magnitude in special cases. In the general case a nominal figure of two to four or more times the performance of the 3000CX is potentially available on the Series II.

Acknowledgments

Thanks go to Chris Wilson and Mas Suto for their efforts in preparing scripts and conducting the many hours of measurement in such a timely fashion.

References

1. J. Hawkes, "TEPE, Time-Sharing Event Performance Evaluator User Guide," Hewlett-Packard Data Systems Division Report, June 9, 1975.
2. N. Mack and L.E. Shar, "A Supersystem for Basic Timesharing," Hewlett-Packard Journal, December 1974.
3. N. Benwell, "Benchmarking Computer Evaluation and Measurement," Hemisphere Publishing Corporation, 1975.

Clifford A. Jager

Cliff Jager graduated from the University of California at Berkeley in 1958. He holds a BA degree in economics, but his career focus has been computer systems. He's done systems programming, BASIC timesharing development, and performance-measurement development and consulting. Since joining HP in 1973 he's been doing performance measurements on the HP 3000 and now manages that effort. Cliff is married, has three children, and lives in Castro Valley, California, just a few miles from Oakland, his birthplace.

Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, California 94304

HEWLETT-PACKARD JOURNAL

AUGUST 1976 Volume 27 • Number 12

Technical Information from the Laboratories of Hewlett-Packard Company

Hewlett-Packard S.A., CH-1217 Meyrin 2 Geneva, Switzerland
Yokogawa-Hewlett-Packard Ltd., Shibuya-Ku Tokyo 151 Japan

Editorial Director • Howard L. Roberts
Managing Editor • Richard P. Dolan
Art Director, Photographer • Arvid A. Danielson
Illustrator • Sue M. Perez
Administrative Services, Typography • Anne S. LoPresti
European Production Manager • Michel Foglia

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

CHANGE OF ADDRESS: To change your address or delete your name from our mailing list please send us your old address label (it peels off). Send changes to Hewlett-Packard Journal, 1501 Page Mill Road, Palo Alto, California 94304 U.S.A. Allow 60 days.