Experience using a time-shared multi-programming system with dynamic address relocation hardware

by R. W. O'NEILL
IBM Watson Research Center
Yorktown Heights, New York

INTRODUCTION

The IBM Research M44 computer is an experimental machine which was installed in the Thomas J. Watson, Sr. Research Center, in Yorktown Heights, New York, in November of 1964. The machine is an extensively modified IBM 7044, a binary single-address fixed-word-length computer with a CPU cycle time of 2 micro-seconds.

The machine was designed by Research personnel with two goals in mind. First, there was a need for a large-scale computer which could be approached at the hardware and software system levels for various kinds of experimental activities. Second, it became important to implement a paging machine and measure its performance under a variety of workloads.

The M44 has become a full computer system which is time-shared by users in the building and around the country. It is a multi-programmed system which is also used to do batch processing. There are a number of experimental software systems which are implemented and play an important role in the work of various departments of the building. And there are several special purpose hardware devices attached through the direct data channels of the machine to facilitate certain other projects.

Among the more interesting applications are:
- a project in the automated design of computers.
- a computer operating system study.
- a project relating to the design and use of terminals in an interacting environment.
- the utilization of special hardware to test monolithic circuits.
- a hardware/software complex which controls traffic in one of the tubes of the Lincoln Tunnel.
- a collection of activities, mainly under an ARPA contract, in the use of such software systems as IPL5, LISP, and SNOBOL.

There is in addition much day-to-day work of a conventional kind, and, what is perhaps the most interesting aspect of the project, the measuring and evaluation of the entire complex of hardware and software.

The intent of this paper is to present the results, to date, of the measuring and evaluating activity. A necessary part of such a presentation is a description of what is being measured. We believe that the characteristics of this paging machine, are translatable, with scaling, to other paging machines and should therefore be of some interest to persons working in this area.

M44 hardware

The M44 computer is a highly modified version of the IBM 7044. It is a single address computer operating with 21 bit addresses, with the instruction format, index registers, and the instruction counter modified to handle 21 bit addresses. It has 32,768 words of 2μsec computational store, plus an additional 196,588 words of 8μsec computational store. The M44 has 3 modes of operation which are not found on the standard 7044: Problem/Supervisor mode, Location Test/no Location Test mode, and Mapping/no Mapping mode. The Problem/Supervisor mode capability is the rather conventional facility of reserving certain instructions, notably I/O and mode changing instructions, for execution only in the supervisor mode. The Location Test Mode in conjunction with the Location Test Register provide a convenient debugging tool which can be used to selectively halt the execution of a program when reference is made to any specified storage cell. The Mapping mode provides a dynamic address relocation mechanism which may be used to solve the problems of storage allocation in a multi-programmed environment.
The mapping device of the M44 computer consists of a 16,384 word 2μsec memory and its associated logic. It operates in the following way—the 21 bit addresses generated in the M44 CPU are broken into 2 parts, say the leading 14 bits and the least significant 7 bits. (See Figure 1). The leading 14 bits are then used to address the 16K mapping memory directly. A 14 bit field of the mapping memory word addressed is then concatenated with the 7 bit field of the logical or unmapped address to create a physical address which is now used to access the core memory. This is the basic mapping procedure. However, there are some refinements to the procedure implemented on the M44. First by concatenating the leading bits of the unmapped address with the contents of a fixed register, called the ID register, it is possible to have the same logical address map into distinct physical addresses without changing the contents of the mapping memory. (See Figure 2). Only the ID register must be changed. Of course since the 16K mapping memory requires a 14 bit field to address it, the sum of the leading bit field and the ID field lengths must always be 14. Therefore one might choose 10 and 4 bit field lengths respectively. The reason for calling this additional register the ID register is probably already apparent. By changing it appropriately when switching the CPU between virtual machines it can be used by MOS to distinguish between the pages of core assigned to distinct 44X’s. The second refinement is required due to the fact that although 2 million distinct logical addresses can be generated ($2^{21}$) only 229,376 words of physical core are available on the M44. Therefore one bit of a separate field (called the status bit field) of the addressed mapping memory word is interrogated by the hardware to verify that this is an address which is currently mapped into a physical address of the M44. If it is not currently mapped, a trap to the supervisory program is generated. The supervisory program can then take appropriate action, for example it might initiate a transfer of the logical page from the back-up store (the 1301 II disk) to a physical core location. Upon completion of this transfer it could modify the mapping word accordingly. Because of the ready availability of the status bit field for interrogation at every memory reference, these bits are used to implement a hierarchy of storage protection spheres in the M44 and are also used for data gathering to assist the experimental use of the M44. A complete listing of the status bits is found in Figure 3. This particular hardware implementation of the address translation function contains two drawbacks which would not be acceptable on a production system. The number of virtual machines (44X’s) which can be identified is a function of the page size, and the 2μ sec mapping time is added to every memory cycle. Since the fundamental translation function is unchanged, for the purpose of experimentation these drawbacks are more than compensated for by the ease of construction of the experimental system.

The I/O and storage configuration of the M44 is shown in Figure 4.

44X (virtual machine) description

One can imagine a more or less ideal computer, or virtual machine closely related to the M44. It has two million words of core. It has in effect a separate channel for every I/O unit. It has an instruction set which differs from that of the M44 because it has no mapping device and no problem mode. It does take traps and interrupt for a variety of purposes relevant to the kinds of virtual hardware which constitute the virtual machine. The definition of the virtual machine is kept at the level of constructable, conventional,
$S_0$ In/Out bit, 1 indicates the block is in core and mapping is allowed, 0 forces a mapping device trap.

$S_1$ Locked In bit, 0 indicates the block is locked in and cannot be referenced, 1 allows references.

$S_2$ Read-Only bit, 1 indicates only fetch-type references are allowed, 0 allows either read or write references.

$S_3$ Reference bit, this bit is set to 1 by the hardware whenever a successful map is made using this block, the bit is reset by program.

$S_4$ Active bit, this bit is set to 1 by the hardware whenever a store type reference is made to this block, this bit is reset by program.

$S_5$ I/E cycle bit, this bit is set to 1 by the hardware whenever I-time (instruction fetch) references are made to this block, bit is reset by program.

$S_6$ Conditional Protection bit, 1 indicates only instructions executed from a "privileged" block are allowed to make stored type references, 0 indicates no conditional protection.

$S_7$ Privileged bit, instructions executed from this block are privileged, to store, see $S_6$.

$S_8$ Transfer Protection, 1 indicates only transfers of control into this block will be allowed if they come from "transfer privileged" blocks.

$S_9$ Transfer Privileged bit, instructions executed from this block are allowed to transfer to transfer protected blocks, see $S_8$.

Figure 3

44X operating system

The 44X Operating System contains a collection of program processors similar to those found in most operating systems. Specifically it includes:

- FORTRAN IV compiler
- SYMBOLIC ASSEMBLY PROGRAM
- BINARY LOADER
- FILE MAINTENANCE SUB-SYSTEM
- COMMAND LANGUAGE SUB-SYSTEM
- DEBUGGING AND CONTROL LANGUAGE SUB-SYSTEMS

In most respects these programs are conventional processors found in most computing systems. They are unique only in their storage management. All of these sub-systems are permanently resident in the virtual 44X execution store. The collection of programs making up the 44X Operating System occupy less than 65,536 words of the 2 million word execution store of the 44X. Consequently the core resident system does not significantly restrict the execution store available to the problem programmer. Because of the generous size of the execution store, problems of intermediate storage management have disappeared from these programs. All of the 44X Operating System has been made Read-Only code by separating the modifiable instructions from the straight line code and executing them using the XEC instruction. Note that while this allows the same copy of the 44X Operating System to be used by many 44X's simultaneously, it does not allow recursion within a single 44X. This is not re-entrant code.

MOS, the M44 modular operating system

The Modular Operating System (MOS) is the program which honors the 44X machine definition on the hardware machine, the M44. MOS is a modular operating system so that many of its design parameters can be easily modified to facilitate system experimentation.

MOS is basically a set of interrupt handling routines whose logical inter-connections are created by the data upon which they operate. These data are for the most part maintained in a series of queues, the most significant of which is the joblist, or CPU queue. This is a list of the States Of Machine (SOM's) of the virtual machines. An SOM contains all essential information about the state of a virtual machine, such as the contents of the AC, MQ, XR's, etc. Each SOM also contains an indicator of the virtual machine's availability for processing, i.e. whether or not it can
use the CPU. The joblist is serviced by the dispatcher routine (DISP) which is executed whenever there is any change of status in the joblist. The dispatcher searches the list, starting at the top, for the first available SOM and then turns the M44 CPU over to that job. Thus there is a strict priority inherent in the ordering of the joblist. The joblist is divided into three sections: the highest priority is the operating system service routines, next the terminal oriented 44X's, and finally non-terminal oriented 44X's (sometimes referred to as background jobs). In addition to these sections of the joblist, there is at the bottom of the list an operating system routine called IDLE which is always ready to run and keeps track of unused CPU time. Ordering of jobs within the joblist may easily be changed dynamically in response to changing requirements. One of the mechanisms which changes the joblist ordering is the time-sharing algorithm.

Time-sharing is controlled by the interval timer subsection of MOS by the following procedure. A timer interrupt transfers control to the interval timer subsection of MOS, after the state of machine has been saved in the SOM entry of the job which was interrupted. The interval timer subsection then changes the priorities of one or more jobs in the joblist, initializes the interval timer for some interval, and then transfers control to the dispatcher routine. The method by which the priorities are changed, and the time interval chosen is not restricted, and therefore the time-sharing algorithm is quite easily changed. Note that there is no direct connection between the operation of the time-sharing algorithm and the storage allocation.

The mapping device interrupt handling routine provides dynamic allocation of the real core store to meet the demands of the virtual machines. The 44X demands are signalled by mapping traps. The mapping device routine (OLSR) handles these storage requests in the following manner. After saving the state of the machine in the interrupted 44X's SOM, control is given to the mapping device routine. The trap is an indication that a virtual machine is attempting to reference a logical address in a page not currently in the M44 core store. Since the mapping device operates upon logical pages not individual words, OLSR must effect the input of the logical page containing the word being referenced if the 44X is to continue processing. OLSR has access to a directory of the core image location (disk address) of all logical pages belonging to 44X's. This provides the location of the page to be transferred into the M44 core. The choice of a core input location generally involves the overwriting of a page in core belonging to some 44X (it may or may not belong to the 44X for whom the new page is being input). The proper choice of the page to be overwritten is critical to the efficiency of the M44/44X system, since poor choices significantly increase the number of disk to core transfers. This choice is accomplished by the replacement algorithm (REPL) which is quite easily changed and it is one of the primary areas of experimentation within the M44/44X system. The page chosen for replacement may or may not require saving before overwriting, depending upon whether it has been changed since its last input. This is indicated by the altered status bit of its mapping memory word. If required, OLSR arranges for its output before it is overwritten. In any case the interrupted 44X cannot continue processing until the page referenced is in the M44 core. Therefore the SOM of the interrupted 44X is made unavailable (not ready to use the CPU) until the exchange of pages is completed and the mapping device is updated. This automatically inhibits the DISP routine from giving control to a 44X which cannot be processed.

In general two methods of 44X creation are used: those 44X's created to process on-line users requests and those created to process standard stack jobs. Note that the resultant 44X's created are not in any way incompatible and can be distinguished only by the presence or absence of an on-line terminal. The method by which the 44X's are created differs although the resulting 44X may not. The creation of a 44X to handle the processing of an on-line users request is started whenever the M44 receives an attention interrupt from a terminal which is not assigned to an existing 44X. In response to the interrupt a minimal 44X is created, i.e., 2 million words of core store including the read/only systems programs and the on-line user terminal. An SOM is created and added at an appropriate priority level in the joblist. The initial SOM instruction counter contents will contain the transfer address of the 44X systems program used to deal with on-line users. Thus when the 44X created is given control by DISP the 44X system will communicate with the on-line user and further interaction between the 44X and the on-line user will be under the control of the 44X system program designed to deal with on-line users. In the event that the on-line user requests the use of additional I/O units (tape, disk, another terminal, etc.) the 44X system program can request the assignment of additional I/O units for this 44X using a well-defined set of 44X instructions. On the current system, MOS will only honor requests of this type when they are executed from the 44X locations known to contain the 44X system programs. However, this convention is not an absolute requirement of this system organization, but without
it 44X program bugs could demand excessive numbers of I/O units and therefore slow down the operation of the system. It is possible that requests for additional 44X components for a particular 44X cannot be met immediately. In this event control is returned to the 44X system program at a fixed location, so that the on-line users program may give the user the option of waiting for their availability, proceeding with some other jobs, or discontinuing his run entirely.

44X's are created to process stacked jobs whenever MOS is informed by the machine operator that background work is ready to be processed. It is expected that as the system develops the operator will provide a backlog of work for the system and MOS will create 44X's to process this work when its load indicates that extra processing capacity is available.

At the time of the creation of a 44X the logical address space to be used by the 44X must be given storage area somewhere in the computing facility. Due to the nature of its use (essentially random, by pages) the only practical storage device in this system is the 1301 disk. However, the assignment of storage space for a 44X's complete logical addressing capability is not practical. The 1301 disk has a capacity of approximately 9 million 36 bit words. If the complete 44X virtual memory were kept on the disk, only four 44X's could be accommodated at any time. This is too few 44X's. Furthermore in any on-line multi-access system it is necessary to have storage for long time storage of user's private files, and this demand must also be accommodated on the 1301 in this system. Therefore the complete 44X virtual memory (2,097,152 words) is not automatically assigned storage space on the disk for the highest 65,536 addresses. In fact the assigned disk tracks are the same for all 44Xs; the read-only 44X system. Another 16,384 words of private storage is assigned for the lowest 16,384 addresses at the time of the 44X creation. This may be thought of as the minimum 44X virtual store claim. At any time during the processing of the 44X more virtual storage space may be claimed. It is possible that a request for more virtual address space cannot be honored, because insufficient disk space is available. In this event the 44X user is given the option of continuing with the space he already has, or halting processing for resumption at some future time when his requests can be met. If this happens too frequently it is an indication that the system is out of balance and more disk (or drum) storage is required.

44X's are destroyed by MOS in response to a 44X instruction requesting self-destruction (POOF). Since this operation will not leave any trace of the 44X which executes it, MOS will only honor this instruction when it is given from a fixed location known to be part of the 44X system control program.

It has previously been stated that certain modules of MOS are replaceable and are changed for experimental purposes. The following modules are debugged and have been used with MOS for measurement purposes.

**Time-sharing algorithm 1**

The first time-sharing algorithm implemented as an MOS module is a very simple one, but it has also proven effective for this type of system organization. This time-sharing algorithm shares the priority implied by the ordering of the CPU queue on a time basis. It does this by reordering the CPU queue at the end of each time quantum. Since the response time of the 44X's with terminals is of importance, only the middle section of the CPU queue is reordered. This section is reordered by removing the top entry and reentering it at the bottom of this section of the queue. This action does not necessarily affect the allocation of the CPU since higher priority jobs (MOS service jobs) may require the CPU both before and after the CPU queue reordering. In contrast to most other time-sharing systems this action has no direct effect upon the real core allocation. Real core is allocated only on demand and in a manner defined by the current replacement algorithm module of MOS. The time quantum which defines the frequency of joblist reordering is a constant in this algorithm.

**FIFO replacement algorithm**

The initial algorithm implemented to allocate the real pages in the M44 was a FIFO (First In, First Out) algorithm. This algorithm was chosen for no other reason than its ultimate simplicity. It provides a benchmark to measure subsequent replacement algorithms against, and it increased the speed with which the initial version of MOS could be debugged. Its effect upon system performance does not recommend it. The FIFO replacement algorithm chooses the real core page to be overwritten by finding the page which has been in the real core for the longest period of time. In practice this means it picks pages to be replaced in a round-robin fashion among the pages allocated to virtual machine pages. This algorithm is extremely crude and has been known to make the worst possible choice of pages for replacement. It has overwritten the page containing the instruction causing a mapping trap, in order to bring its data into the real memory. Of course when the data page is in real core it is now necessary to replace another page to bring the instruction back to use it! Fortunately it cannot get into a loop of this type.
AR replacement algorithm

On the basis of extensive simulator studies of 7094 programs (reported in the IBM Systems Journal, Vol 5, No. 2, “A Study of Replacement Algorithms for a Virtual Storage Computer,” by L. A. Belady) the AR replacement algorithm was the most promising page replacement algorithm which was studied. This algorithm required some additional hardware on the M44 and consequently was not implemented immediately. The “AR box” hardware contains a bit for every page in the real core which is used for paging. Each bit is set by the hardware whenever a reference is made to its corresponding real page. The complete set of bits is scanned by the hardware after every memory reference; if all of the bits are set they are all reset. This mechanism guarantees that there is always at least one unset reference bit. The AR replacement algorithm chooses its replacement page from the set of pages which have not been referenced since the last reset. Hopefully the lack of reference in the recent past indicates that these pages will not be required in the near future.

Biased replacement algorithm

In addition to all the replacement algorithms which have been described, tests have been made using a biased version of the FIFO replacement algorithm. The biased version differs from the non-biased replacement algorithm by the addition of a restraint. A favored 44X is chosen and the replacement algorithm is biased in favor of the chosen 44X by rejecting for replacement any pages belonging to the favored 44X. The designation of the 44X for which the system is biased is changed on a regular basis. The biased FIFO algorithm picks a new 44X after every second cycle of complete search the real memory. This implies that the rotation of memory preference is a function of the page replacement rate and the percentage of pages belonging to the favored 44X.

M44/44X system performance measurements

Measurement and evaluation tests have been made for two distinct purposes: comparison of virtual machine organization and more conventional system organizations; and to evaluate the effect of various parameters on the performance of virtual machines. The effect of various virtual machine parameters will be discussed first.

To simplify the comparison of test results all testing was done using only the 8μsec memory. Since all virtual machine execution is done in the mapping mode, and the 2μsec mapping cycle is not overlapped, the effective cycle time of the 44X is 10μsec. Significant parameters for the interpretation of all test results are: type of work load processed; page size used; time-sharing quantum, if the time-sharing algorithm was used; real core size; and the degree of multi-programming i.e. the number of virtual machines processed concurrently. For all the test runs which will be discussed these parameters will be specified, but it is very important to keep in mind that the results apply only for the particular combination of parameters used. Unless otherwise specified all loads are processed using tape for Sysin and Sysout.

Effect of page size

The virtual machine organization separates the logical memory addressed by problem programs into pages of a fixed size for the purpose of storage allocation. This implies that when any particular word of the virtual memory is required for execution, a full page of the logical memory must be present in the real execution store. Because of the regularity of most programs some of the words in the page will probably be required in the immediate future for execution; however not all of the words will be required. Those words which must be in the execution store due to the arbitrary break-up of programs into pages, but are not required for execution, represent wastage of real core. (Note that non paged system organizations do not avoid similar wastage of execution store, since storage claims generally exceed those necessary for execution.) The amount of core required for this type of wastage depends upon two factors, the organization of the program and the size of the pages being used. Figure 5 illustrates the effect of different page sizes upon the amount of execution store required to fully accommodate the needs of two different job loads. For the load marked A-taped (10 small FORTRAN compilations and loads), the required core goes from 20,992 words to 73,728 words. With 8,192 word pages it is clear that at least 62,736 words of execution store is wasted since the same program never references more than 20,992 words when 256 word pages are used. Some of the core required even with 256 word pages is unused, but how much is not clear from the data at hand. That the wastage caused by large page sizes is dependent upon the load is clear from the core requirements of the job market FTT (FORTRAN compilations and executions, used to debug the 44X FORTRAN compiler). The increase in core required for this job is a small percent of the total requirements as the page size is increased.

The previous curves represented the total core required for the complete processing of some jobs. However because of the paging mechanism it is not necessary to devote this much core to a job at any one
Paging provides an automatic overlay mechanism when less than complete program requirements are available. Figure 6 shows the effect on throughput of reductions in the amount of core devoted to a particular job's requirements. As the amount of core is reduced, more paging activity is necessary and performance suffers. The performance degradation is slight up to the point when not enough pages are available in the M44 to fully accommodate the basic segments of the job load. At that point performance suffers drastically. The degradation in performance small job (PT-A) is more sudden than that for the job which requires more execution store to meet its complete requirements. This occurs because the size of core necessary to completely accommodate the major loops of the programs does not vary nearly as much as their total execution store requirements. Large programs tend to consist of more segments, not just bigger segments. It is worth mentioning that even at the worst performance point shown for the larger job (FTT), which is with 8,192 words of real core, its processing time is no more than that required to process this job on the 7094 II under 1BSYS. The change in storage management technique has gained quite a bit in this case.

**Effect of multi-programming**

Performance of the M44/44X system may or may not be improved by multi-programmed operation depending upon a number of factors. Significant among these factors are: the amount of real core available to the system, the number of autonomous devices in the system (i.e. channels and processing units), and the meshing of the requirements of programs being multiprogrammed. Figure 7 illustrates the effect of the level of multi-programming (i.e. the number of virtual machines processed concurrently) upon CPU efficiency. Efficiency increases with increasing load up to the point where competition for pages in the real core between virtual machines creates excessive amounts of page turning activity. With excessive page turning performance suffers drastically, since the typical state of the virtual machines becomes one of "page wait." This property has several important implications for system design. The operating system should continuously monitor its operation, and automatically reduce the level of multi-programming, by setting work aside, if it detects an overload condition. Second, the system must have adequate execution store if the full benefits of multi-programming are to be realized.

**Effects of time-sharing**

The time-sharing algorithm module of MOS was primarily intended to insure an adequate response time...
on those 44X's with terminals attached. However, provision was made to allow the time-sharing of non-terminal 44X's when turnaround time was of great importance. Experiments using this capability demonstrated that time-sharing often has significant advantages even when turnaround and response times are not important. Figure 8 shows the total time required to process the same load with various amounts of real core, both with and without the time-sharing algorithm in operation. In this example with more 64K of memory the total processing time is less with time-sharing than without. When less than 64K of memory is available the system would function more efficiently with time-sharing and a reduced level of multi-programming although this is not illustrated by Figure 8. The explanation for this behavior might be summed up by saying “the more you stir the pot, the more mixed up it becomes.” In more technical terms what appears to be happening is a demonstration of the fact that a program which has run on the CPU for a time period t, without any I/O request, has a lower probability of requiring I/O than a program which has not used the CPU. Therefore frequent switching of programs tends to insure that the overall balance between demands for CPU and channels is better. Some testing has been done with time-slice quanta other than 1/10th second. The effect of varying the slice in the range 1/60th to 1 second is negligible. The importance of the time-sharing algorithm on a system of this type is not great, since it has no direct effect upon the storage allocation.

Comparison of the M44/44X system with other computing systems is inevitably a comparison of the total systems including both hardware and software packages. Therefore comparisons of this type have a somewhat limited significance. Unfortunately there does not exist any satisfactory alternative to this type of comparison. One possible way to evaluate systems is to compare the time, or cost, required to perform some “typical” job. This we have done, but it must be emphasized that in real life no particular job is “typical,” and therefore the results are biased by the jobs chosen.
Comparisons have been between the 7094 and the M44/44X using three distinct test jobs: Product Test Tape A, B, and C. Product Test tape A contains 10 relatively small (about 200 statements each) FORTRAN IV source language main programs each with several sub-programs. Product Test tape B consists of 4 FORTRAN source language programs for compilation and execution, with all four of the object programs very heavily oriented toward numerical calculations with no problem program I/O at all. Product Test tape C contains 6 FORTRAN programs, each of which performs conversion for output, and writing of 10,000 records on private tapes. In order to properly evaluate the comparative timings of the 7094 II and the M44/44X system it is necessary to bear in mind that the memory cycle times are 1.4 μsec and 10μsec respectively, and the tape speeds are 90KC and 60KC respectively.

Test tape A requires 6 minutes and 24 seconds to compile and execute on the 7094 II, with no loading or linking of the object decks. On the M44/44X system, utilizing multi-programming at the level of 3, with 1/10 second time-slicing, all problems on Test tape A can be compiled, loaded, and linked in 50 seconds. This is clearly a load better suited to the M44/44X system organization. The most significant gains of the M44/44X system accrue from the in core compilation, the elimination of separate loadings of the compiler for each compilation, and multi-programming. All three of these are directly related to storage management.

Product Test tape B requires 5 minutes to compile and execute on the 7094 II and 27 minutes 43 seconds to compile and execute on the M44/44X system, with no multi-programming. It is interesting to note that the ratio of completion times is almost exactly the ratio of the memory speeds of the systems. No multi-programming was done for this comparison on the M44/44X since nothing can be gained by multi-programming several jobs all of which require only use of the CPU. This type of job may be advantageously multi-programmed with more I/O oriented jobs. When this is done, the comparison between systems can no longer be made as any assignment of increased speed to particular jobs of the multi-programmed mix is purely arbitrary.

Product Test Tape tape C requires 15 minutes to compile and execute on the 7094 II and 17 and ½ minutes on the M44/44X; when the M44/44X is multi-programmed to the level of 2 (the number of overlapped tape channels on the M44). Better thru-put can be achieved from the M44/44X system by combining the B and C tapes. For example, two C tapes and one B tape run together require 57 minutes and 31 seconds to complete on the M44/44X. With all three jobs started simultaneously, one C tape finished after 30 minutes, the other C tape after 37 minutes and the B tape after 57 minutes and 31 seconds. Since the last 20 minutes were not multi-programmed, further gains could be expected from further mixing of the load.

**On-line time-sharing experience**

In addition to the controlled experiments run with background jobs considerable experience has been gained while the M44/44X system was providing computing service to on-line users. The results of this experience are not amenable to the analysis which can be given to the results of controlled experiments. Changes in system parameters are often not as important in changing the system characteristics as are the changes in the behavior and composition of the individual terminal users. However the results of monitoring on-line usage do have value. Some of the data monitored have shown remarkable consistency in the face of both system changes and user changes. Correlations between properties of system behavior are also of interest for system analysis. Finally some feeling for the behavior of the users and the varying responsiveness of the system can be of value even though it can not be precisely quantified. Because the system was designed from the start as an experiment in system design many statistics are gathered by the system. At the end of every system up-time period these statistics are reduced to readable form and saved.

**Idle time significance**

The measure of performance has been processing time, i.e., the time required for the system to completely process the job load. For simple types of measurement this is a satisfactory measure, but it is inadequate for comparisons of performance on different job loads. Some jobs are inherently short jobs and others are long. A more satisfactory measure is the efficiency with which the hardware components are used. A somewhat indirect, but effective, measure of CPU usage is the percent of time spent in the MOS "idle loop." The "idle loop" time is wasted CPU time since it is precisely that time when the system has no meaningful function to perform on the CPU for either problem program or control program requirements. This occurs when all virtual machines are in wait status, awaiting completion of I/O. The I/O may be either problem program I/O or paging I/O. The "idle loop" times should not be used to make direct comparisons with non-paging systems, since idle times may be attributable to either paging induced
activity or to problem program I/O which is non-overlapped.

The latter contribution to the idle times is part of the inherent nature of the job being processed rather than an attribute of paging systems. In addition some of the paging I/O (ex. initial program load) shows up on other systems although not in the form of paging. The time required for the housekeeping activities of the control program (MOS) was not measured since it is difficult to distinguish between paging induced housekeeping, and control program execution of necessary problem program functions. With these reservations in mind the “idle loop” percentages can be viewed as a comparative measure of the efficiency of various configurations of the M44/44X system.

Figure 9 shows the page replacement rate and the percentage of CPU utilization (non-idle time) on a minute by minute basis during a prime-shift time-sharing session. The figures in the left-hand column are the actual number of replacements during a minute, times the page size divided by 256. This method of normalization allows comparisons of paging rates even though different page sizes were used. The relationship between the paging rate and the CPU utilization, while not completely regular is apparent. The CPU utilization appears to be inversely proportional to the paging rate. However when the paging rate goes down this clear relationship disappears. Figure 10 represents the next 44 minutes of the same session as shown in Figure 9. In Figure 10 the CPU utilization appears to be independent of the paging rate, although the page rate varies rather widely. During the time-sharing operation of the M44/44X system Figure 10 represents a more typical state of the system than Figure 9. The importance of these two curves is that under overload conditions the paging rate can get out of hand and reduce the overall performance of the system dramatically. However a properly designed system can protect itself against such performance degradation. It is necessary that the system monitor its own performance and when an overload is detected it can defer some of its load to a period of less demand.

The wide variations in the paging rate, even when monitored on a minute by minute basis, illustrates the difficulty of using the average paging rate as a measure of system performance. In addition to the wide variation in paging rate there is a difference in the load upon I/O channels due to paging, depending upon the type of pages being brought into the real core. The page being replaced may or may not require saving depending upon whether it has been modified since its last input from a secondary storage device. Similarly a page being called for may or may not require input from a secondary storage device depending upon whether this logical page was ever used by the virtual machine before. A page which has never been referenced by the virtual machine cannot have data meaningful to the 44X in it, in which case any page in
the real core will do. Because of these two possibilities the amount of I/O required to complete a page replacement is of interest when deciding upon secondary storage facilities for a paging system. During monitored time-sharing with the M44/44X system the number of pages transmitted to and from the secondary storage device has varied from 1.1 to 1.5 page transmissions per replacement. The higher figures in general representing sessions during which the average page replacement rate was higher than normal.

The section of the 44X virtual store which contains the 44X system programs is read-only and consequently a single copy in the real memory can be shared by all of the 44X currently using the system. The implementation of MOS provides for such sharing of the virtual system pages. A count is maintained which reflects the average number of virtual machines which have referenced each system page during its lifetime in the real core. During time-sharing sessions when up to 16 users are allowed on the system this shared ratio has varied from 3.5 to 5.0 when the FIFO replacement algorithm is being used. On the basis of this data it is clear that the ability to share frequently used system programs significantly improves the utilization of the execution store of a paging system.

SUMMARY
Experience to date with the M44/44X system has demonstrated the feasibility of the concept of dynamic relocation using paging hardware. Some improvements in the implemented system have become obvious, but even the initial implementation appears to have significant advantages for some types of processing. The controlled system experimentation done on this system represents a useful first step in advancing system design from its present state as an art to an engineering science. Large amounts of data have been, and are being gathered by this system relevant to paging system behavior. The significance of much of these data is not yet clear. Much remains to be done.