THE EFFECTS OF INPUT/OUTPUT ACTIVITY ON THE AVERAGE INSTRUCTION EXECUTION TIME OF A REAL-TIME COMPUTER SYSTEM

Salvatore C. Catania
Univac, Systems Design and Simulation Group
Data Processing Division
Blue Bell, Pennsylvania

Abstract

The paper describes the results of a GPSS simulation of a real-time computer system undertaken to determine the effects of specific types of Input/Output (I/O) activity on the average instruction execution time (AIET) of the system's central processor. The types of I/O activities studied are communications I/O and references to mass storage devices. Various memory bank configurations and different memory bank referencing schemes are considered in an attempt to minimize the effects of the I/O activity on the AIET.

1. INTRODUCTION

Many third generation real-time computer systems employ various hardware devices which make it possible for the system to handle communications input/output (I/O) activity while simultaneously executing instructions in the Central Processor Unit (CPU). In effect these devices remove the responsibility for handling the I/O activity from the CPU in order to allow the CPU to devote more of its time to instruction execution. Theoretically, it is possible for several operations to take place concurrently within the system without interferring with each other. However, this simultaneous activity is possible only if the CPU and the I/O activity handlers are referencing different memory banks contained within the computer system. Should any two devices attempt to reference the same storage bank at the same time, in most cases, the bank will service the requests on a priority basis, and one of the devices will have to wait until the other device has been serviced before it receives its requested memory cycle. The question which naturally arises concerning such a simultaneous access system is: What is the frequency of occurrence of these conflicts at the memory banks and what is their accumulative effect upon system performance? If a system designer were to configure a real-time system and compute the response time of an incoming transaction on the basis that (1) all required processing activity occurs sequentially and (2) all possible activity that can be overlapped with other activity occurs in an overlapped manner, he would find that two completely different response times are obtained. The actual response time would like somewhere between the two extremes computed.

In many cases, finding the upper and lower bounds of the response time would be sufficient. In many other cases, the range between these two extremes is too large and must be narrowed. In an attempt to measure the effectiveness of simultaneous processing in a real-time system, a model was developed which simulates a computer with simultaneous access capabilities and generates statistics which give information relating to the frequency of occurrence of the memory bank conflicts and their effect upon overall system performance. The statistics give information concerning (1) the average wait time at a memory bank, (2) the effective transfer rate of I/O activity, and (3) the average instruction execution time (AIET) of the CPU. These results can be used to help predict, with greater accuracy, the performance of such a system in a real-time application.

2. DESCRIPTION OF THE SYSTEM MODELED

The basic computer configuration to be studied is shown in Figure No. 1. It consists of a CPU, two Input/Output Modules (IOM) to handle I/O data transfers, and N memory banks. The Input/Output Modules actually function as small independent

processors that direct the I/O data transfer operations and provide the system's I/O channels. It is important to note that the Input/Output modules function independently of the CPU.

In order to accomplish the objectives of the study, the manner in which the hardware items simulated reference the main storage banks and transferred data has to be included in the model.

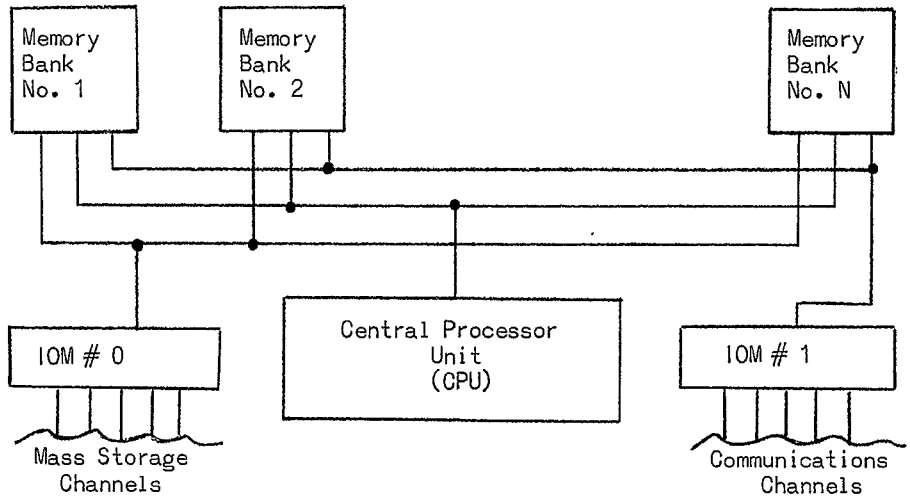

Figure No. 1

Real-Time System Configuration

The IOM channels are assumed to be connected to mass storage media or communications lines. Data being transferred over any channel is assumed to be transferred in either of the following modes:

(1) Internally Specified Indexing (ISI) mode
(2) Externally Specified Indexing (ESI) mode

The ISI mode of operation permits each word of data transferred over a channel to be transferred in one storage cycle. Mass storage media make data transfers in the ISI mode. The ESI mode of operation is used when more than one data communications line is connected to the same I/O channel. In this mode of operation, Buffer Control Words (BCW) are stored in main storage and an index value that indicates the location of the BCW is specified by the external device (communications terminal) as the data is transferred. The ESI mode allows each I/O channel to operate as a multiplexer servicing a large number of communications lines which are transferring data concurrently. ESI data transfers require more than one memory cycle to accomplish the data transfers. The study discussed here assumes that four memory cycles are required per ESI character transferred. In both the ISI and the ESI mode, once the transfer of a block of data is initiated, no CPU cycles are required to effect the individual character or word transfers of the data block. In the study, the CPU is assumed to be continuously executing instructions from a pre-determined instruction mix, and both ISI and ESI data transfers are occurring in a random manner.

Insofar as IOM references to main storage are concerned, the process is very simple and will be explained later. The operation of the CPU is somewhat more complicated and will be explained in detail.

As stated above, the CPU is assumed to be continuously executing instructions during the analysis. In general, the types of instructions being executed by the CPU are governed by the application being studied.

Two important instruction mixes are frequently used for analysis purposes in studies of this type--The Gibson Mix and the Scientific Mix. The applicability of these mixes to given applications will not be discussed here. Suffice it to say that a probability density function specifying the types of instructions to be executed and the probability of occurrence of a given type is a necessary input to the model. Regarding the individual instructions in the mix, the only area of interest is the number of memory cycles requested by a given instruction and the time, relative to the instruction's start, that the memory cycles are requested. Two typical computer instructions are shown in Figures 2A and 2B.

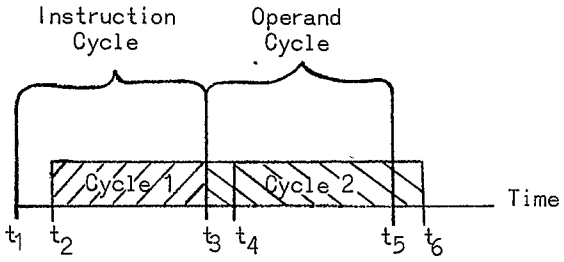

Figure 2A

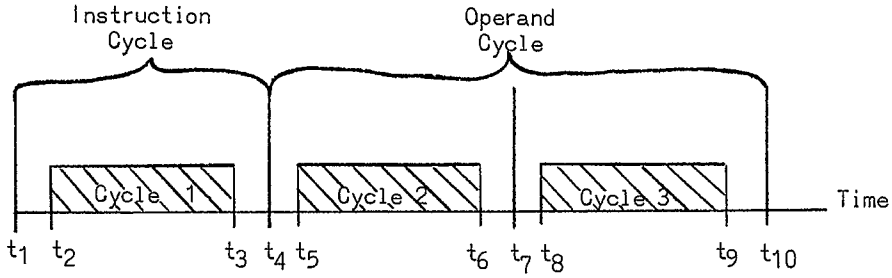

Figure 2B

CPU Instruction Types

The instruction depicted in Figure 2A is a type which requests two memory cycles. These cycles are requested with no time interval between the requests. The instruction illustrated in Figure 2B requests three memory cycles, and there is a time interval between these requests. With the instruction of Figure 2A, the CPU begins to execute the instruction at time t_1. At t_2, the first memory cycle is requested from one of the system memory banks. If no wait time develops at the memory bank granting the access, the CPU finishes the instruction cycle of the instruction at time t_3 and immediately begins the operand cycle. At t_4 the first memory cycle is completed and the second cycle begins. If no memory wait occurs, the CPU finishes the operand cycle at t_5, and the memory bank being referenced is busy until t_6 at which time it finishes the second memory cycle. The memory cycles requested during this instruction execution may or may not be requested from the same memory bank in the system. Should the bank that is to grant the request be busy at either t_2 or t_4, the time required to execute the instruction will be delayed an amount equal to the time spent waiting for the memory access. An instruction of the type shown in Figure 2A with a memory wait included is given in Figure 3.

At time t_4 the memory bank which is to be referenced for the second memory cycle is busy. This busy condition exists until time t'_4 when the second cycle begins. As a result of this memory wait, the times noted by t_5 and t_6 are delayed a time interval equal to t'_4-t_4. The memory wait has, in effect, increased the execution time of this instruction an amount equal to t'_4-t_4. The effect of repeated memory waits similar to the one described above is obvious, i.e., the average time required to execute an instruction becomes longer than the time which would be required were no memory conflicts to arise. This lengthening of the average instruction execution time would naturally result in longer program execution times and higher CPU utilization--both undesirable results.

Typically, medium and large scale computers have an instruction repertoire which includes in excess of one hundred instructions. When these one hundred or so instructions are placed into type categories determined by time-memory request figures such as shown in Figure 2, usually less than ten to twelve different instruction type categories result. It is very simple, then, to model the computer system's entire instruction repertoire since only the dozen or so different instruction types need be modeled.

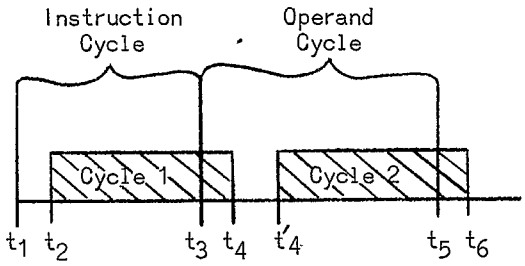

Figure 3

Delayed Instruction

Memory cycle requests made to the memory banks by the Input/Output Modules are very straightforward. In the ISI mode one cycle is requested per word transferred. Should a memory wait develop, the IOM simply waits to obtain its required access. In the ESI mode, multiple memory accesses are requested (four in the study discussed here). The requests are made one at a time, i.e., the IOM cannot request four consecutive cycles. It must request the cycles one at a time and contend with the other requesting devices for each cycle. Should a wait develop for any one of the memory cycle requests, the succeeding requests are all delayed. The effect of memory waits developing during ESI or ISI data transfers is a slowing down of the IOM transfer rates. That is, data is transferred at a rate lower than the hardware speeds alone would dictate. This slower data transfer rate could have some adverse effects upon the computer system (communications data is lost, drums are caused to go through extra revolutions in order to transfer data, etc.). Such side effects are, needless to say, extremely undesirable and should be avoided whenever possible.

Hardware considerations and the utilization of the system memory cycle requesting devices alone do not determine the number of conflicts that will occur at the memory banks. The system software and core allocation--core location of the ISI data, ESI data, worker programs, and the Executive System--are important considerations and have a pronounced effect upon system performance. Since detailed information of this type is usually not known until a time much later than the time at which studies of this type are usually conducted, an attempt was made to find the limits of the variation in the results by making different assumptions concerning the operation of the system software and core allocation. Basically, the system was studied by assuming:

Case A - ISI data, ESI data, and programs are scattered throughout core. For any given data block transfer or a communications request, selection of the bank to or from which the data is to be transferred is made by assuming that it could go to any bank in the system with an equal probability. Once a bank is selected for a transfer, all of the data in that particular transfer goes to the bank selected. For each instruction executed by the CPU, a bank is selected from the banks present on an equi-probable basis. Once a bank is selected for an instruction, both the instruction and operand cycles requested their access from the selected bank.

Case B - Specific banks are designated to contain only ISI data. Other banks contain only ESI data, and the remainder of the banks contain the programs. ESI and ISI transfers are made to or from their designated banks only. During the execution of an instruction, the CPU makes its instruction cycle request to one of the program banks. The operand cycle, if the instruction has one, requests its memory cycle from either an ISI or ESI designated memory bank. For realism groups of instructions - between ten and one hundred - make their operand cycle request to the same memory bank.

It is the feeling of the writer that Case A will cause a maximum number of memory conflicts and Case B a minimum. Needless to say, a program and a core allocation scheme could be devised which would result in more memory conflicts than will occur in Case A. The feeling here is that, if such a situation does arise, it is really the result of careless programming. Such a case will

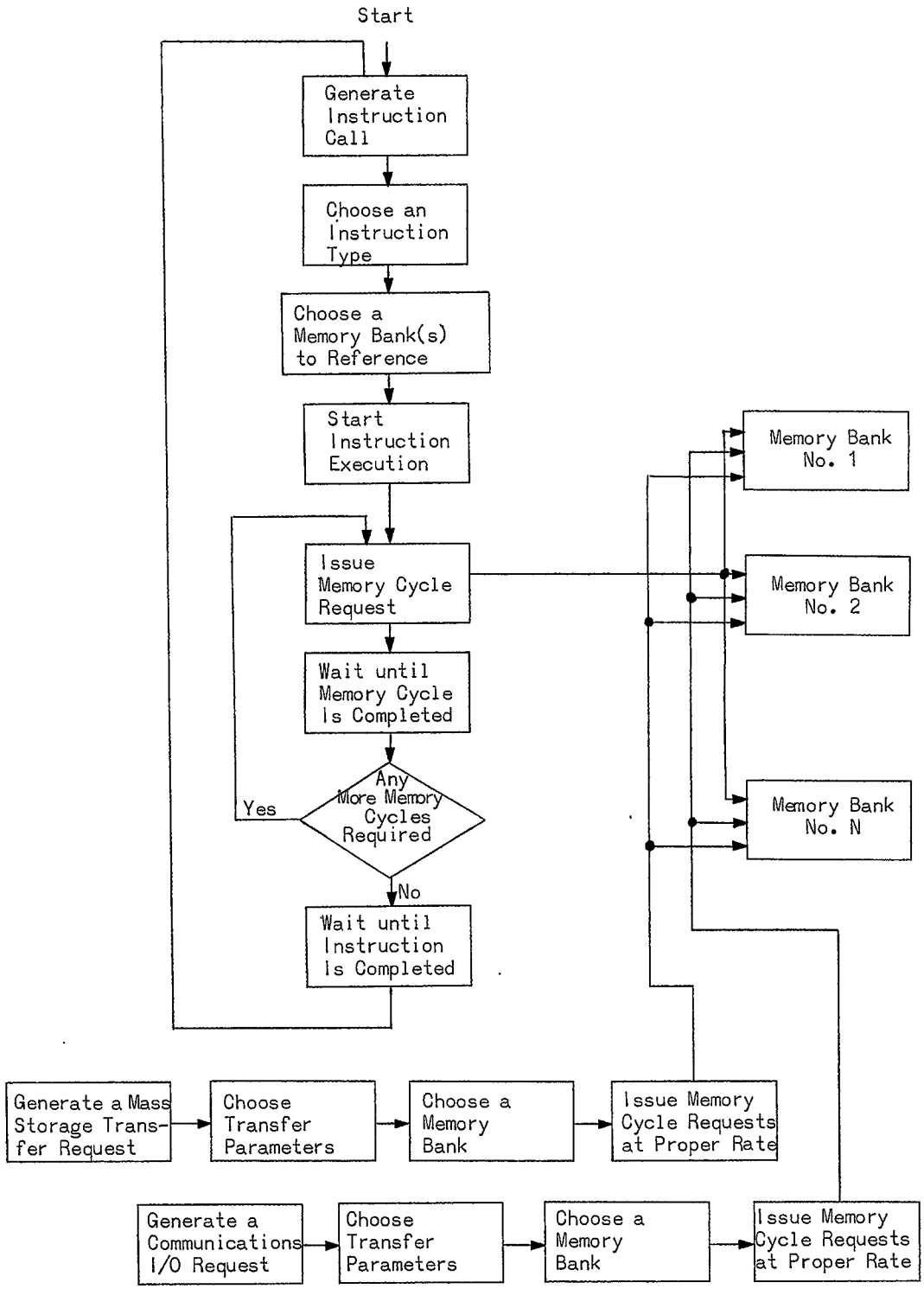

Figure 4

Simulation Model Flowchart

not be considered in the results presented here. The terminology "worst case" and "best case" must be interpreted, however, with the above comments in mind.

In all of the plots to be presented the IOM utilization was taken as the independent parameter. Since it was possible--actually quite probable--that the utilization of the system Input/Output modules would not be equal, the utilization given is the average IOM utilization.

3. SIMULATION MODEL

The general flow of the simulation model is shown in Figure No. 4. Three devices capable of requesting memory cycles from the N memory banks are simulated. The simulated CPU begins the execution of an instruction immediately upon the completion of the previous instruction. The initiation of a new instruction causes the instruction type to be selected from an instructions type distribution. The distribution is varied in accordance with the application being studied. Once the instruction type is selected, the CPU is "seized", and the storage cycles are requested from the memory banks at a time determined by the instruction being simulated. Any memory conflicts that occur cause the instruction's execution to be halted until the memory request is satisfied. Operating independently of the CPU and also requesting memory cycles is an IOM which is transferring simulated data blocks from mass storage devices. The time between block transfers from the various devices is selected from a Poisson inter-arrival distribution. The number of words transferred per block is drawn from a normal distribution. The rate at which memory cycles are requested for a given transfer, i.e., the mass storage device's data transfer rate, is a function of the mass storage devices being simulated. Several mass storage devices can be active and transferring data at different rates concurrently. The maximum number of devices which can be active at any one time is governed by the hardware characteristics of the IOM being simulated, which for all practical purposes may be simply thought of as a multiplexer. The other simulated device which requests memory cycles from the banks is another IOM. This IOM is transferring communications I/O. The inter-arrival time of the communications requests is selected from a Poisson distribution also, and the number of characters transferred per I/O request is selected from a normal distribution. The rate at which memory cycles are demanded per I/O request is governed by the transmission speed of the line being simulated. As is the case with the mass storage data block transfers, several communications I/O requests can be active simultaneously. Conflicts arise at the memory banks whenever more than one device requests a memory cycle at a given memory bank at the same time. Each bank processes its request queue on a first-in-first-out basis. All of the delays which result from the conflicts are tabulated in the simulation output.

GPSS was selected as the simulation language since the program was relatively small and the logical model was well suited for implementation in the language. The longer run times that resulted from using GPSS were more than compensated for by the savings in programming time. In the simulation model the CPU, the Input/Output modules, and the Memory Banks were treated as facilities which were seized and released by the program transactions. A transaction in the program could represent either an instruction, a mass storage data block transfer, a communications I/O request, a memory cycle request, or a word or character to be transferred. The location of a transaction in the program, naturally, determined its physical meaning.

The GPSS simulation program was about 200 cards in length, and approximately two man weeks were required to program and debug the simulation model once a logical model was at hand. Program execution time varied between two and fifteen minutes on a ®UNIVAC 1108 System. The number of instructions executed by the simulated CPU varied between 5K and 50K. The high number of instructions executions and the longer run time were necessary to achieve stability in the output statistics when low IOM utilization was being simulated. In order to determine when stability had been achieved, the output statistics were printed every 5000 simulated CPU instructions. Stability was said to have been achieved whenever the statistics varied less than 1%.

4. RESULTS

The results obtained from the study of Case A of Section 2 will be discussed first. Figure No. 5 shows a family of curves giving the increase in the average instruction execution time as a function of the average IOM utilization. The family of curves was generated by running the model with 2, 3 and 4 memory banks in the system.

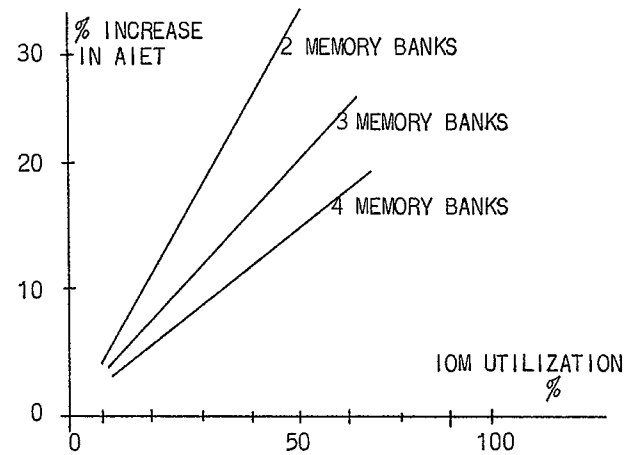

Figure No. 5

AIET Vs. IOM Utilization-Case A

®Registered Trademark of Sperry Rand Corporation

In the figure the average instruction execution time is plotted as a percentage increase over the value that would result with no IOM activity. The curves show that the average instruction execution time increases in all cases with increasing IOM activity, and the rate of increase becomes higher as the number of memory banks is decreased. When the IOM activity is low (less than 10%), little difference is noted in the average instruction execution time as the number of memory banks is decreased. This is an important result since one of the primary objectives of the study was to determine when the addition of memory banks could be used as a means to speed up the system.

Perhaps a word should be said here about memory system design choices in a system of the type being discussed. A memory bank in our system can have a capacity of either 16K or 32K words. If it has been determined that 64K words are required to contain The Executive System, all applications programs, and the data, the system designer may obtain this amount of core in either two or four memory banks. There is quite a difference in cost between these two memory systems. The results of Figure No. 5, then, gives some guidelines which may be used in selecting the number of banks to be placed on the system. If a particular computer system being designed has an average IOM utilization of 10%, employing two banks instead of four causes only an additional 2.5% increase in the AIET. This increase will manifest itself mainly in a 2.5% increase in program execution time and CPU utilization. Since most real time systems are designed with CPU utilizations in the neighborhood of 20-50%, this slight increase in utilization will have no real adverse effect upon overall system performance. Economy would then dictate that the system should be configured with two memory banks instead of four.

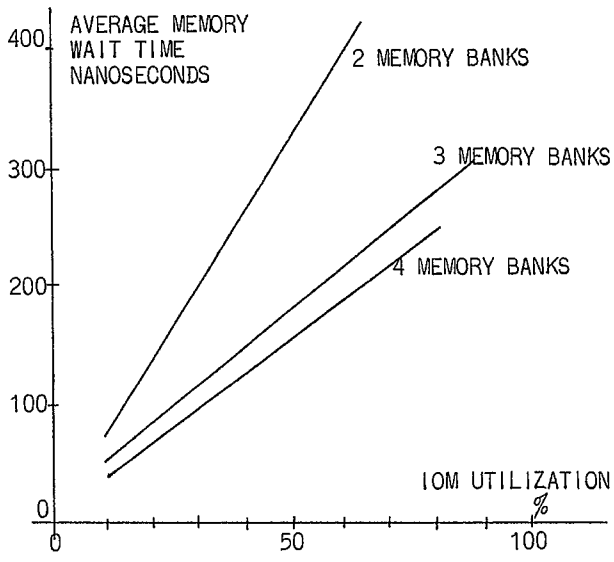

Figure No. 6

Average Memory Wait Time Vs. IOM Utilization - Case A

Figure No. 6 shows a family of curves which give the Average Memory Wait Time as a function of IOM activity for 2, 3, and 4 memory banks. As with Figure No. 5, for low IOM activity rates, there is little difference in the wait time between the two and four bank systems. This result could have been anticipated since the average instruction execution time results of Figure No. 5 merely reflect the effects of the memory wait times. Figure No. 7 shows a family of curves which give the IOM word and character transfer times for ISI and ESI data transfers. With both types of data transfers, the transfer times increase as the utilization rate is increased. The system studied, theoretically, has an ISI word transfer time of .75 microseconds (usec) and an ESI character transfer time of 3.0 usec. An interesting phenomenon can be noted in both families of curves. If any of the curves, either ESI or ISI, are extended, they will not intersect the theoretical transfer time supposedly obtained when there are no memory bank conflicts. Unfortunately, stabilization problems prohibited running of the model for average IOM utilization below 10%. The indication is definitely here, however, that even for low IOM utilizations, the transfer times obtained would be reduced some 10 to 15% below their theoretical value. A further interesting bit of data is obtained from the curves in the high IOM utilization range. Transfer times are seen to increase some 50 to 100% when IOM utilization approaches 70%. This indicates that applications which require utilizations computed to be 70% or so using the theoretical transfer time would more than likely exceed the system transfer capacities because of the reduction in the IOM efficiency.

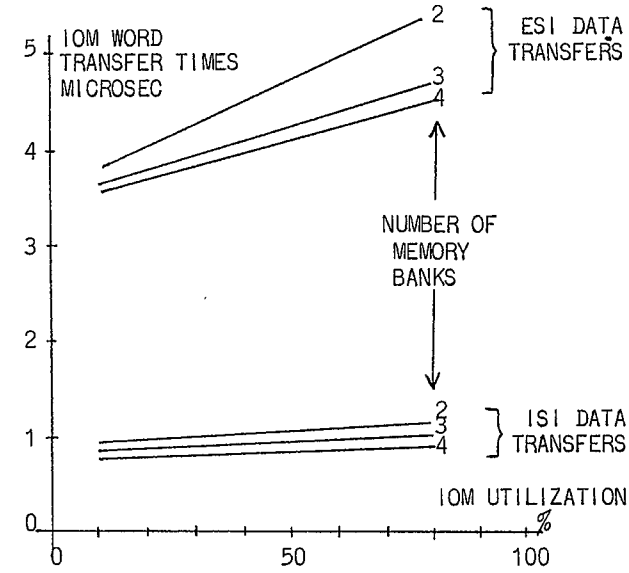

Figure No. 7

IOM Word Transfer Times Vs. IOM Utilization - Case A

Figure No. 8 shows two curves which give the increase in the average instruction execution time versus IOM utilization of 3 memory bank systems of Cases A and B of Section 2. A marked reduction in the increase in the average instruction execution time is noted when the memory allocation scheme of Case B is employed. This reduction in the average instruction execution time is seen to occur at all IOM utilization values. It may be concluded that a definite improvement in program execution speed can be obtained when the software is designed to minimize the number of memory conflicts that occur. Further, the curves indicate the magnitude of the improvement. Comparison of the Case B curve of Figure No. 8 to the 4 memory bank curve of Figure No. 5 shows that a greater improvement in system performance can be obtained by improving the software than may be obtained by simply adding hardware.

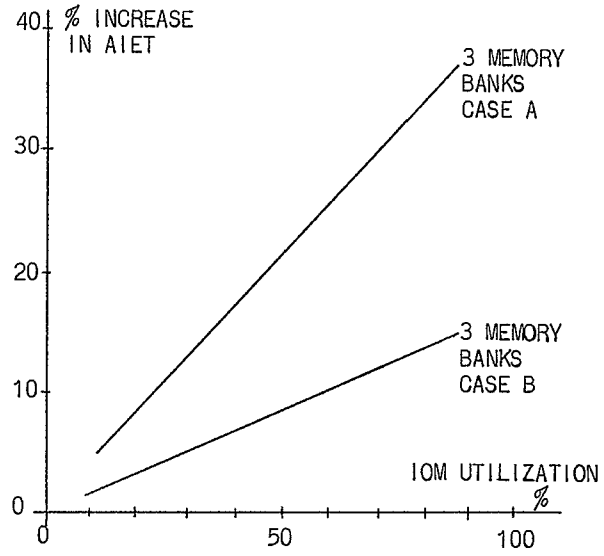

Figure No. 8

AIET Vs. IOM Utilization - Case A, Case B Comparison

Figure No. 9 shows curves giving the average memory wait time versus IOM utilization for 3 bank systems of Cases A and B. A reduction in the average wait time is noted for Case B. The reduction is not as dramatic as the improvement noted in average instruction execution time plots. This is due to the fact that the wait time plotted for Case B is the average of the waits that occur at the two memory banks in which data is stored. These are the only banks where memory conflicts can occur. No waits can occur at the bank(s) which contain the Executive and the applications programs since only the CPU references these banks. Therefore, during the instruction cycle of each instruction executed, no memory wait occurs, and the CPU executes instructions at a greater speed.

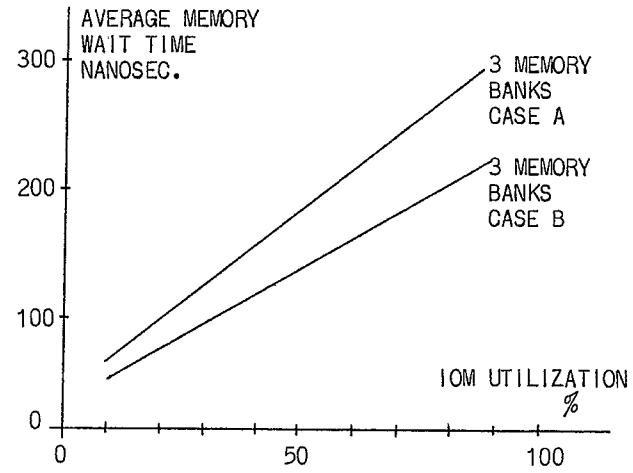

Figure No. 9

Average Memory Wait Time Vs. IOM Utilization
Case A, Case B Comparison

Figure No. 10 shows curves giving the IOM transfer times for 3 bank systems of Cases A and B. A reduction in the transfer times can be seen for both transfer modes in Case B. This is not surprising since the probability of a conflict at a given memory bank is reduced in Case B, i.e., the IOM's cannot conflict with each other and either IOM can only conflict with the CPU during its operand cycle. In Case B the IOM transfer times should be independent of IOM activity. This can be noted in the figure and served as one of the many model validity checks employed in debugging the model.

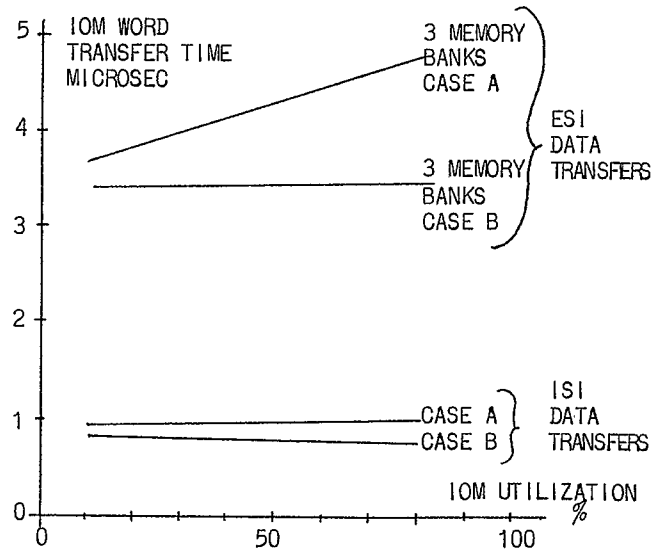

Figure No. 10

IOM Word Transfer Time Vs. IOM Utilization
Case A, Case B Comparison

5. CONCLUSIONS

In an effort to increase the speed of real-time computer systems which handle many I/O requests, many third generation computer systems contain hardware devices which allow simultaneous processing of instructions and I/O requests. The model described in this paper provides a method by which the effectiveness of these devices may be measured in particular applications. The hardware necessary to obtain this simultaneous processing is expensive and should not be employed unless its use can be justified. A model, then, to evaluate such systems can be extremely useful since mathematical techniques - Queueing Theory, etc. - cannot easily be applied to study the problem, and the problem does require study.

The model described here cannot be easily modified to study different computer systems, i.e., the operation of the hardware varies so greatly from system to system that a separate model is required to describe each particular system to be studied. Once a model has been developed for a system, however, it may be re-used many times to evaluate different configurations used in a wide variety of applications. The general format followed in developing this model, however, can be used to develop similar models to study the same problem.

The usefulness of the model can be attested to by the results described in Section 4. Guidelines for selecting system hardware, system limitations in a given application are predicted, and the value of improving software over adding hardware is demonstrated. A model of this type can be a powerful tool in system configuring during a proposal effort since it allows configurations to be quickly evaluated in a particular application.

6. BIOGRAPHY

Sal Catania was born in Philadelphia, Pennsylvania in 1934. He received his A.B. in Mathematics from Temple University in 1961, and his M.S. in Engineering from the University of Pennsylvania in 1968. His professional background includes work in the following areas: analog computer simulation studies of high performance control systems for electro-hydraulic test equipment, hybrid computer methods for performing power spectrum analysis studies, and scientific programming of general engineering and mathematical problems. His work at UNIVAC has been in the analysis and modeling of real-time computer systems.