A SIMULATION OF BUS ARCHITECTURES FOR MULTIPROCESSOR SYSTEMS

Larry L. Wear, Ph.D.
Ron Guilmette
Mark Falash
Department of Computer Science
California State University, Chico
Chico, CA 95929

Abstract

This paper presents the results of the simulation of three different bus architectures used in multiprocessor systems. The TIMESHARED BUS, MULTIBUS, and, CROSS BAR SWITCH configurations were modeled. Activity diagrams were used as a basis to develop the models. Since our models were based upon hypothetical multiprocessor systems, there was no way to compare the models' performance with actual hardware performance. Therefore, independent models were developed in both FORTRAN and Pascal to validate the results. Each model contains a number of functional units, i.e. memory modules (MM), central processing units (CPU's), and input/output processors (IOP's), connected by a particular bus configuration. The results of the simulations are presented as a series of graphs that display average percentage utilization curves for a particular module class, such as CPU's, as a function of some independent variable such as I/O request rate. The results of some of the simulations are compared with similar results published by other authors.

INTRODUCTION

The low price and availability of central processing units have made it economically feasible to consider connecting several such processors together into one system. There have been a number of proposed architectures for interconnecting CPU's, memories, and IOP's. This paper focuses on three of these configurations: the TIMESHARED BUS, MULTIBUS, and the CROSS BAR SWITCH (1,2,3). Models for these three architectures were developed in both FORTRAN and Pascal. A previous paper described the tradeoffs involved in choosing the language in which to develop a model (4). The primary reason for using FORTRAN and Pascal was that they are available on most minicomputers and many microcomputers. Our simulation programs are therefore more transportable than those developed in simulation languages such as GPSS or SIMULA.

Each of our simulation programs has three major sections: an input phase in which parameters are entered into the model, a computational phase which calculates the performance, and an output phase that creates a data file which is used to plot the results. Using FORTRAN and Pascal rather than a simulation language made it relatively easy to develop the input and output portions of the models. However, the computation portion of the simulation was more difficult to write since the primitive operations such as event scheduling were not directly available in either language.

The results of the simulation are shown as a series of plots. The plots typically show the percent ACTIVE, FROZEN, or BLOCKED TIME as a function of a variable such as frequency of I/O requests. Each plot includes a family of curves in order to minimize the number of graphs needed for the results and to allow for easy comparison of related curves. Because of the large number of variables involved, it was not possible to include the results of all of the simulations that we performed in this paper. The MULTIBUS SIMULATION section shows some of the results of our simulations of the multibus architecture with several different parameter settings.

BUS ARCHITECTURE DESCRIPTION

The three architectures are shown in Figures 1 through 3. The TIMESHARED BUS (TSB), which is shown in Figure 1a, has all CPU's, IOP's, and MM's attached to a common bus. This architecture is also called COMMON BUS and is used on many models of the Digital Computer Corporation PDP 11 series of computers where it is called the UNIBUS. It is the most simplistic of the three architectures and the number of modules on the bus can vary without affecting the design of the bus interface. In addition, each module requires only one bus interface. A performance limitation of this type of bus structure is the bus contention problem caused by multiple modules trying to access the bus at the same time.

A relatively simple extension of the TSB architecture has two or more common buses that are shared by all the devices on the system. The multiple TSB system gives increased reliability and throughput as compared to the TSB system. It is however, more

Proceedings of the 1982
Winter Simulation Conference
Highland * Chao * Madrigal, Editors

82CH1844-0/82/0000-0269 $00.75 © 1982 IEEE

expensive since it requires multiple bus interfaces for each module. A multiple TSB system as shown in Figure 1b was simulated by our group.

One possible MULTIBUS configuration is shown in Figure 2. This architecture is also referred to as MULTIPORTED MEMORY and offers some performance improvements over a simple MULTIBUS system. The Modcomp IV/35 uses this type of architecture with a four ported memory system (3). An idealized version of this bus architecture would contain one bus (and one memory port per memory module) for each CPU and IOP in the system. Because each MM must have an interface for each bus, the cost and complexity of this type of architecture can be excessive.

The third architecture modeled, the CROSS BAR SWITCH (CSB), is shown in Figure 3. The Carnegie-Mellon multi-mini-processor system, C.mmp, uses a 16 by 16 cross bar switch to connect memory modules to processors (5). The CSB system is different from the other systems in that the bus must be intelligent (2) rather than passive. This can be both an advantage and a disadvantage. Because a CSB architecture includes intelligent switching components, it may be much more expensive than other architectures. It may also be less reliable because its active components are more likely to fail than passive ones. The primary advantages of the CSB architecture are that it allows parallel transmissions and that each functional unit requires only a single, simple bus interface.

It is beyond the scope of this paper to go into all the advantages and disadvantages of the three architectures. A good discussion of this topic can be found in Enslow's book (2).

PASCAL SIMULATION

Several authors have described how they have used Pascal as a simulation language by adding procedures that give it attributes similar to those of GPSS or SIMULA (6,7). Our Pascal simulation program contains procedures to perform the equivalent of the following standard GPSS block statements:

ADVANCE	LEAVE	PRINT,,B	SPLIT
ASSEMBLE	LOGIC-R	PRIORITY	TERMINATE
ENTER	LOGIC-S	RELEASE	TEST
GATE-LS	LOOP	SEIZE	TRANSFER
GENERATE	MARK	SELECT-NU	

Many GPSS standard blocks were not needed and thus were not implemented. The most notable omissions in the above list are the QUEUE and DEPART blocks which were found to be unnecessary in gathering the desired statistics from our simulations. To facilitate the use of subroutines in the simulation, two non-standard blocks, CALL and RETURN, were implemented. These blocks perform the operations implied by their names and do so more efficiently than the transfer block modes normally used to implement subroutines in GPSS.

The following Standard Numerical Attributes were also implemented as FUNCTIONS in the Pascal simulation program:

SNA	Meaning
M1	transaction mark time
N	total block count
W	current block count
R1	random number generator #1

Additionally, two non-standard numerical attributes were created in the Pascal program to permit certain statistics to be calculated with a high degree of accuracy. These were:

FU	total facility utilization
SU	total (weighted) storage utilization

Normally, the GPSS programmer only has access to the values provided by the above SNA's in an indirect manner. Facility and Storage utilization figures may normally be obtained only in parts per 1000 of total elapsed (simulated) time. The standard SNA's which provide these values are FR and SR respectively. These SNA's provide utilization values with only three significant digits of accuracy. By using floating-point division in Pascal, and the non-standard SNA's (FU and SU), percentage utilization figures may be calculated with precision limited only by the floating-point format used by Pascal.

The Pascal program was very "simulation specific", which made the implementation of most GPSS "control" statements necessary. At the beginning of each independent simulation, the program resets all statistics (as though a CLEAR statement had been processed) and then proceeds to read a new set of simulation control parameters from a parameter file. If the end-of-file has not yet been reached on this file, the program acts as though a START statement had been processed and begins execution of the new simulation. Once the end-of-file is detected on the parameter file, program execution is terminated.

MODEL DESCRIPTION

In our simulations, each functional unit was assumed to be in one of four possible states at any one point in time. The four possible states were: FREE, ACTIVE, BLOCKED, and FROZEN. As can be seen from our activity diagrams (4), not all types of functional units actually spend time in each of the four states (e.g. memory modules are never FROZEN). The FREE state implies that a module is not currently performing any function or waiting for any other module. The ACTIVE state implies that a module is performing some operation and is not waiting for any other module. The BLOCKED state implies that the module has requested the use of another module and that the module is not available. The FROZEN state implies that the module has acquired the use of another module and is waiting for that module to complete a requested operation. The activity diagrams given in (4) show the precise conditions under which module state changes are assumed to occur in our models.

The CPU's in the system are characterized by two parameters: cycle time and instruction mix. Both of these parameters are established by the user prior to execution of an individual simulation. The cycle time (a simple numeric value) is an

independent variable that defines the speed of the processor. The instruction mix is an independent variable with a probability distribution. This distribution describes the probability of occurrence of one, two, and three cycle instructions.

The IOP's are relatively simple devices. They are characterized by their cycle time. This is also a user specifiable independent variable. For simplicity, we chose to run all our simulations with IOP cycle time equal to memory access time.

MM's are characterized by two parameters: access time and cycle time. Memory access time is defined as the time required to transfer the address and data information to a memory module at the beginning of a memory write operation (once the selected MM becomes available to the requesting unit). Memory cycle time is defined as the time required to complete a memory read (once the selected MM becomes available to the requesting unit). This time period consists of three parts: 1) address transfer to memory, 2) read operation cycle, and 3) data transfer back to requesting unit.

Buses have either one or two parameters that govern their operation. The bus cycle time parameter exists in each of the bus architectures we simulated. In addition to the cycle time, for CBS systems, buses have an access time parameter which determines how long a module must wait to be given access to the bus after a request for the bus is made. This is also known as the "switching delay".

An additional parameter that must be specified by the user is the I/O request rate. The user must specify the average and distribution of I/O requests as a function of the number of instructions executed by each processor.

THE MULTIBUS SIMULATION

Although three different architectures were simulated, only the results of the simulation of the MULTIBUS model are presented here because of space limitations. The simulation data for the other architectures are available from the authors. The results shown in the following graphs are for a system with 3 CPU's, 3 IOP's, and 3 MM's. All of the graphs use the average number of CPU instructions executed between I/O requests (per processor) as the abscissa which is scaled in log base 2. All of the plots display average values for all modules of a given type in the simulated system.

The first set of plots, Figures 4a, 4b, and 4c, show the percentage of time the CPU's spent in the ACTIVE, BLOCKED, and FROZEN states. Each graph has three plots that correspond to three different memory cycle times. For these three figures the CPU cycle time was 200ns and the bus cycle time was 20ns. We assumed that I/O transfers were buffered and took place at memory speed and that a bus, once allocated to an IOP, was dedicated to that device for the duration of the block transfer. This meant that the IOP's were never BLOCKED waiting for a bus after the initial request was granted.

The plots for ACTIVE and BLOCKED times display what happens when the IOP's are given priority over the CPU's as they were for this simulation. Since the time spent in the FROZEN state is a direct function of the number of memory requests made, this value increases as the CPU's are allowed to make more requests.

The plots in Figures 5a, 5b, and 5c show the average amount of time the IOP's spent in the ACTIVE, BLOCKED, and FROZEN states. The CPU, BUS, and I/O cycle times were the same as those for Figure 4. Since the IOP's were given priority over the CPU's, they only contend with each other for use of the bus and memory. Figure 5a shows that the ACTIVE time for IOP's is proportional to memory cycle time and Figure 5b shows that the FROZEN time is inversely proportional to the memory cycle time.

The next set of plots, Figures 6a, 6b, and 6c, show how the CPU's operations are affected by the CPU cycle time with a fixed memory cycle time of 200ns. The plot of CPU ACTIVE shows that a slow processor does not have to wait as much for memory as does a fast one; this is what one expects and others have also verified this result. Figure 6b is quite similar to Figure 4b; they both are a result of IOP's being given priority over CPU's. The plot of FROZEN time indicates that for high I/O rates, the CPU cycle time does not significantly affect FROZEN time. However, as the CPU's get more memory cycles, the amount of time they are forced to wait for an operation to complete is increased.

The final two graphs show how memory ACTIVE time is affected by memory cycle time, Figure 7a, and CPU cycle time, Figure 7b. The first figure indicates that for compute bound tasks, i.e. those with relatively low I/O rates, slower memories are more often ACTIVE. The second figure shows that, for high I/O rates, memory ACTIVE is proportional to CPU cycle time.

Figure 4a

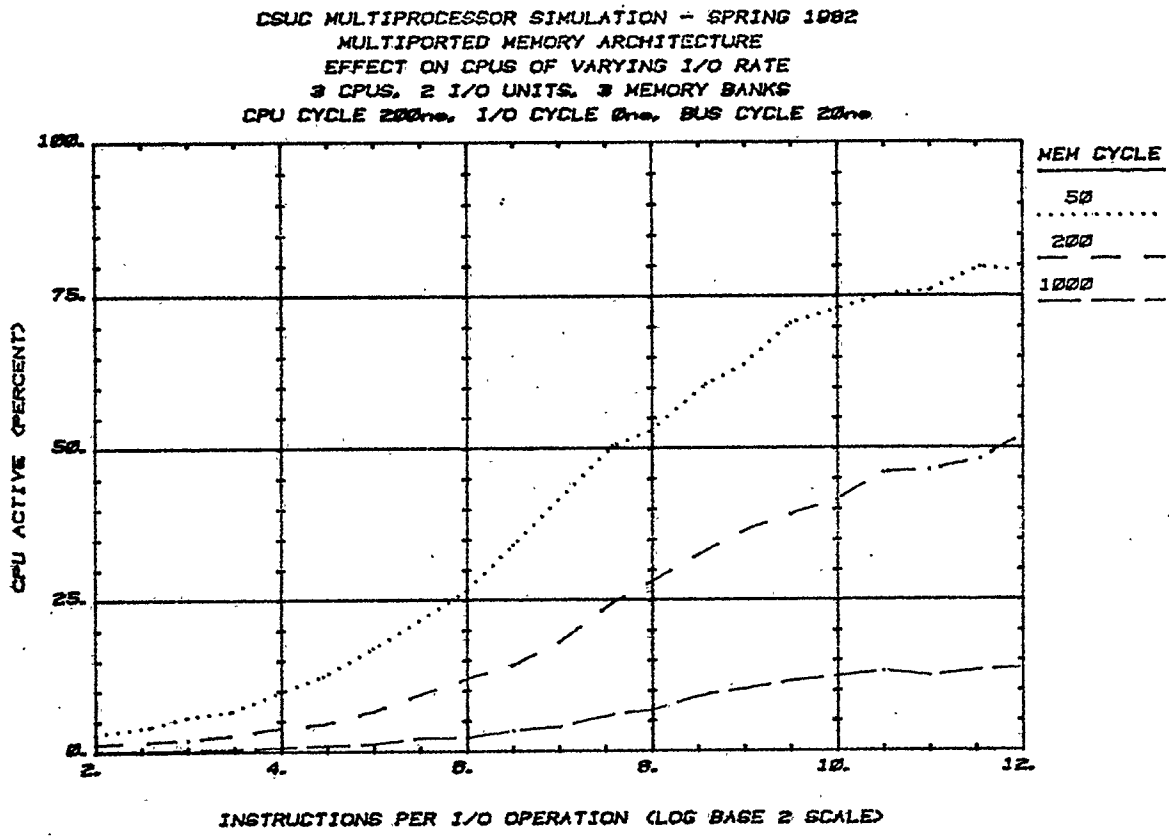


Figure 4b

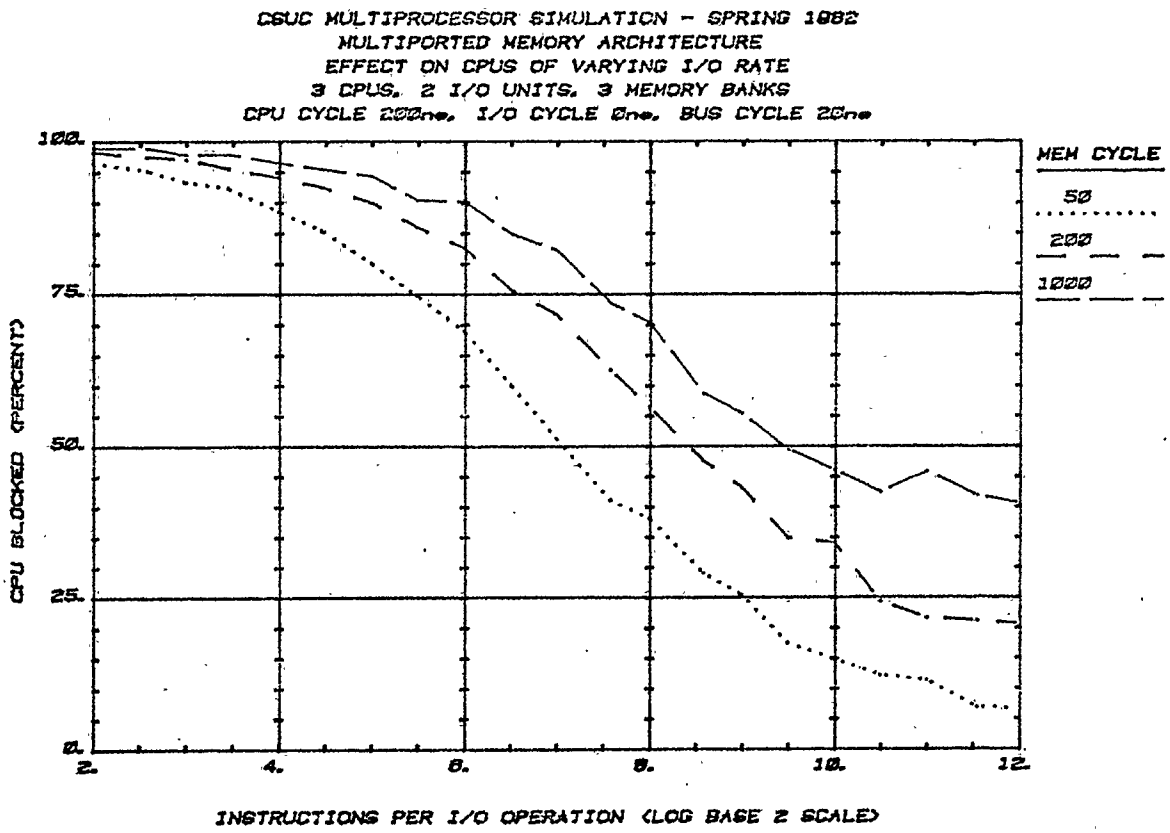


Figure 4c

CSUC MULTIPROCESSOR SIMULATION - SPRING 1982
MULTIPORTED MEMORY ARCHITECTURE
EFFECT ON CPUS OF VARYING I/O RATE
3 CPUS, 2 I/O UNITS, 3 MEMORY BANKS
CPU CYCLE 200ns, I/O CYCLE 0ns, BUS CYCLE 20ns

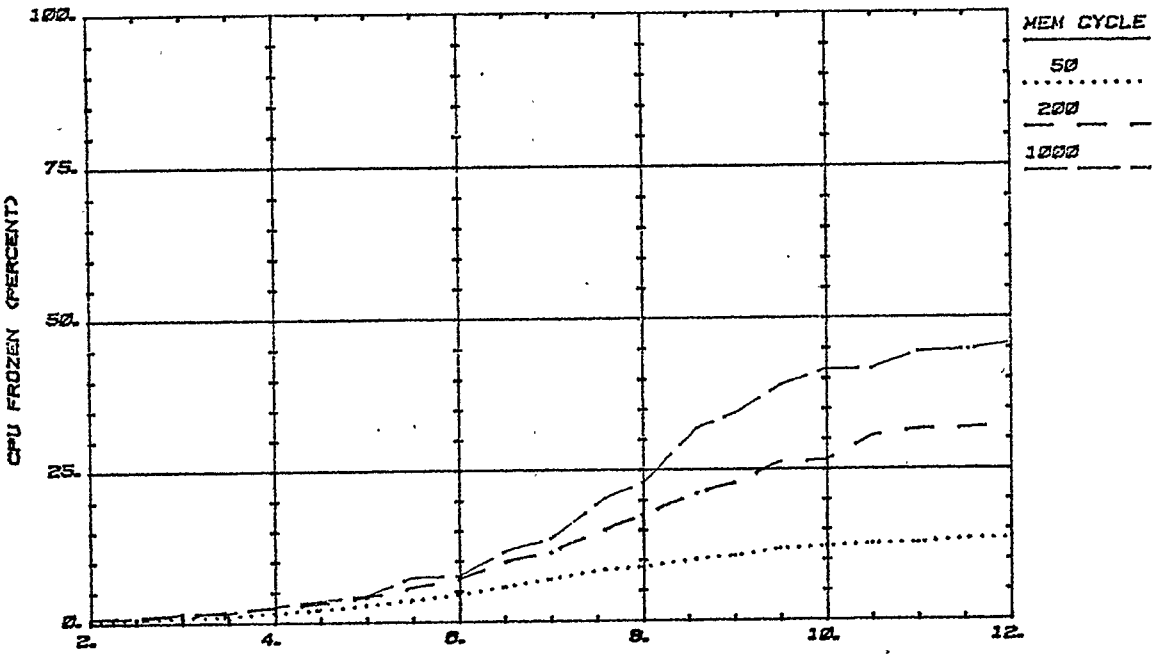


Figure 5a

CSUC MULTIPROCESSOR SIMULATION - SPRING 1982
MULTIPORTED MEMORY ARCHITECTURE
EFFECT ON I/O OF VARYING I/O RATE
3 CPUS, 2 I/O UNITS, 3 MEMORY BANKS
CPU CYCLE 200ns, I/O CYCLE 0ns, BUS CYCLE 20 ns

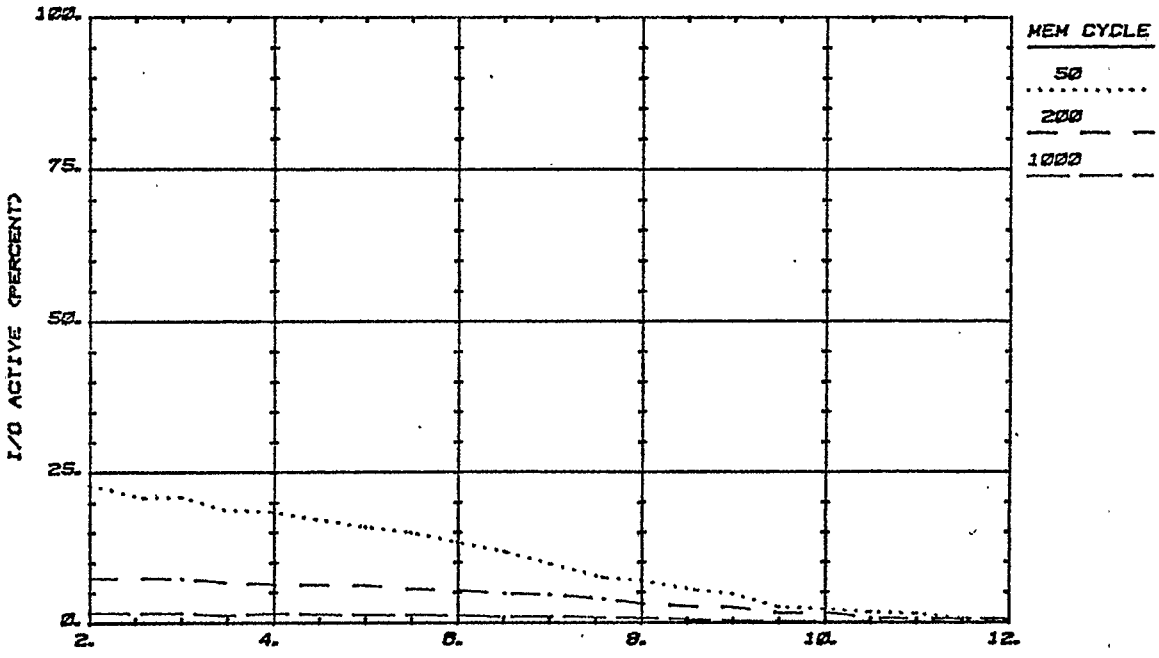


Figure 5b

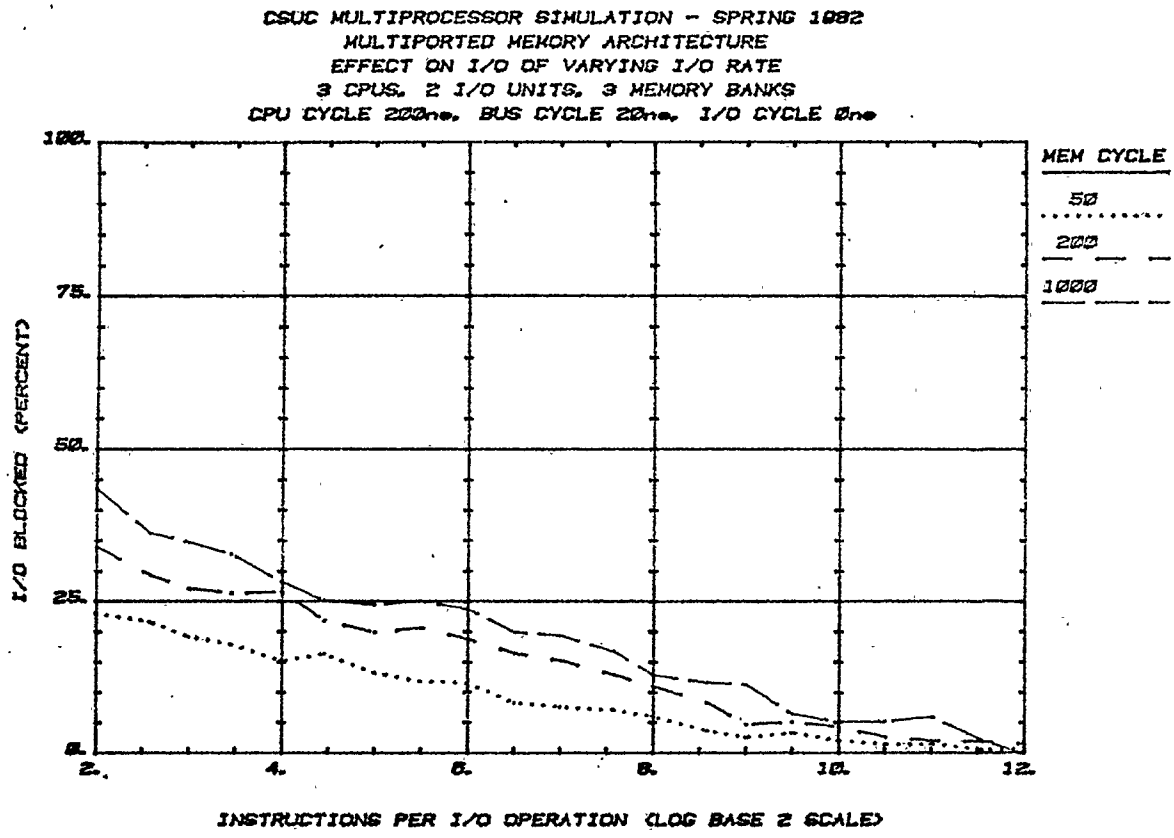


Figure 5c

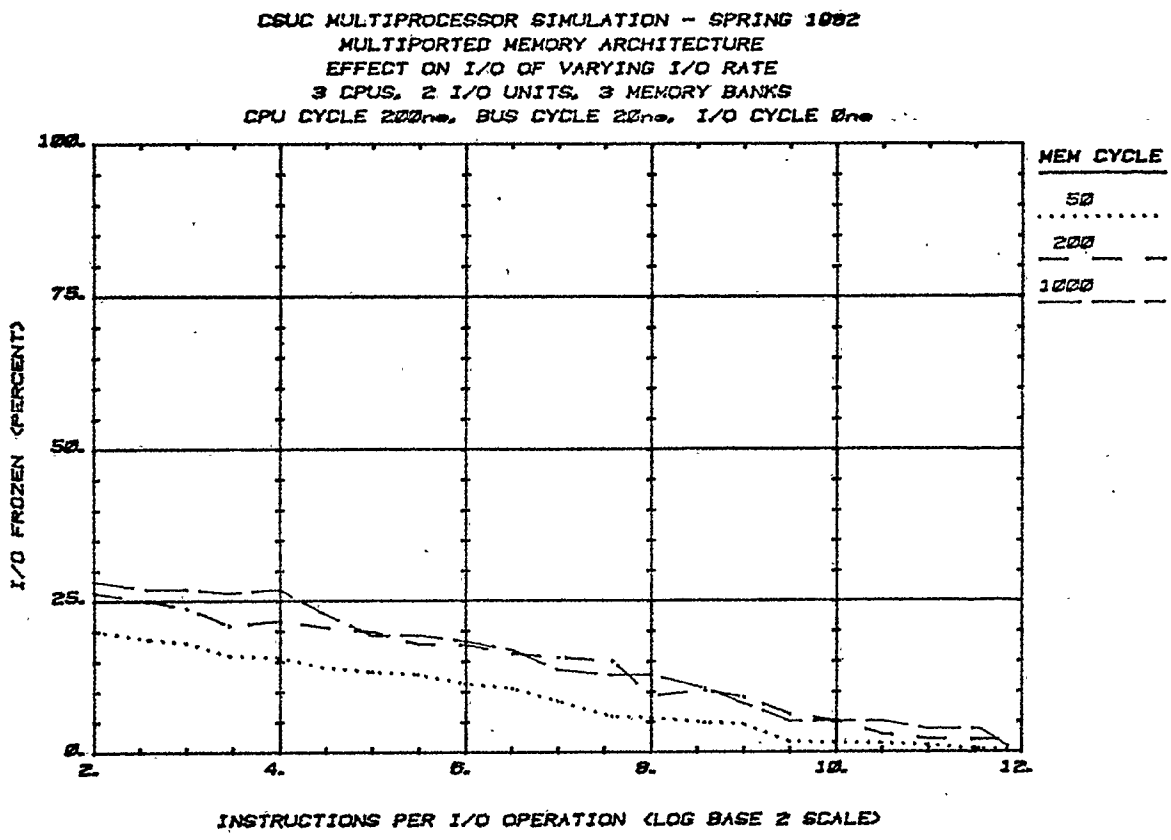


Figure 6a

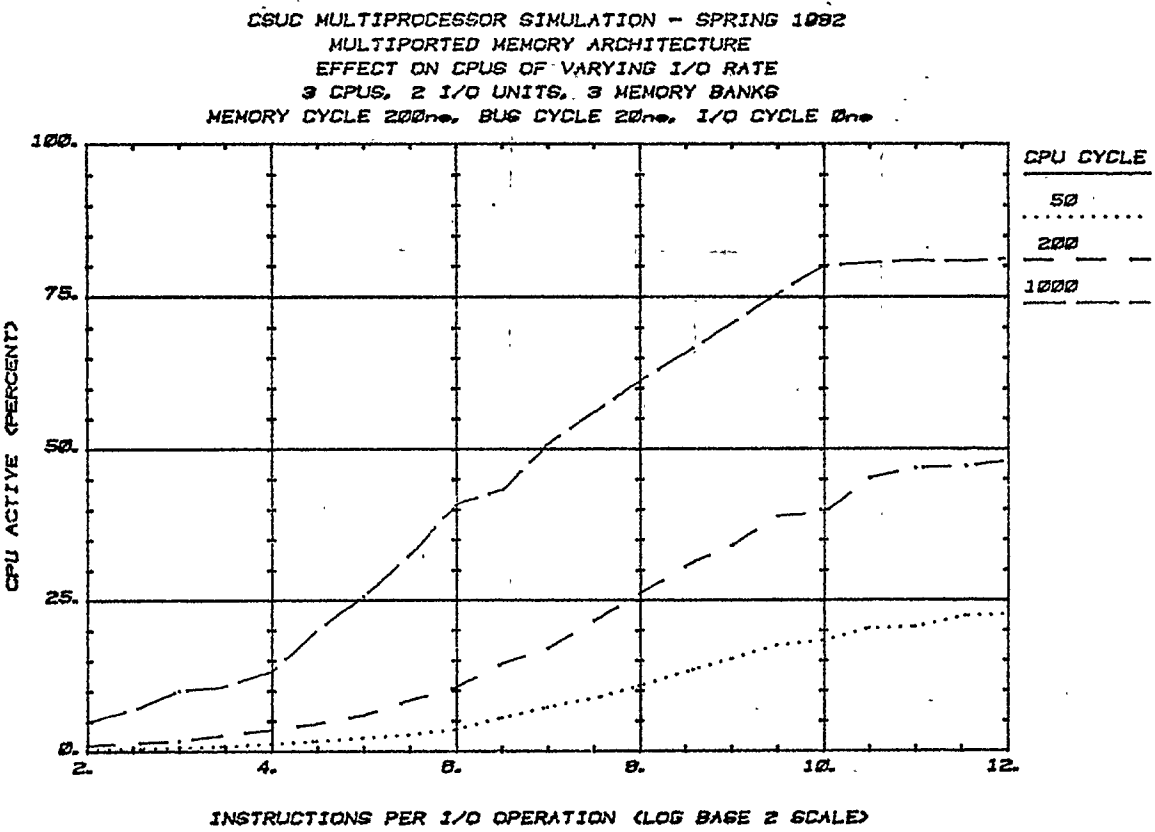


Figure 6b

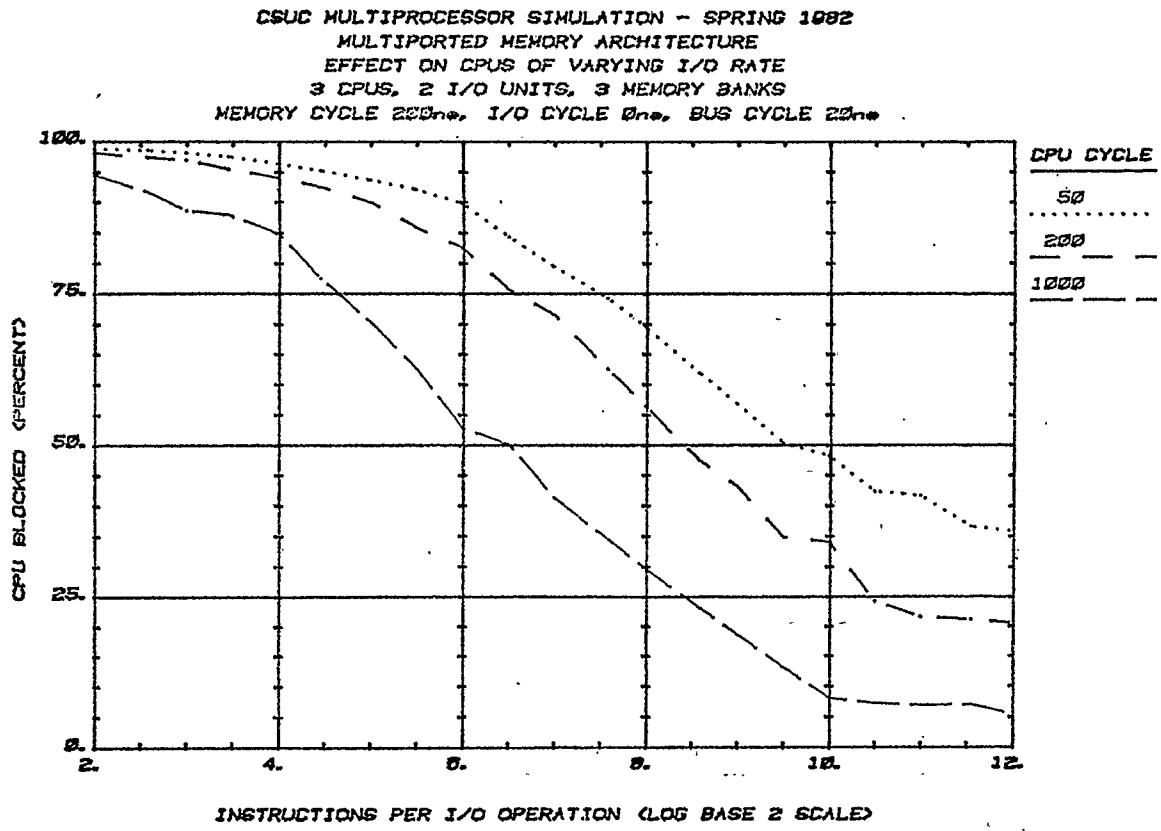


Figure 6c

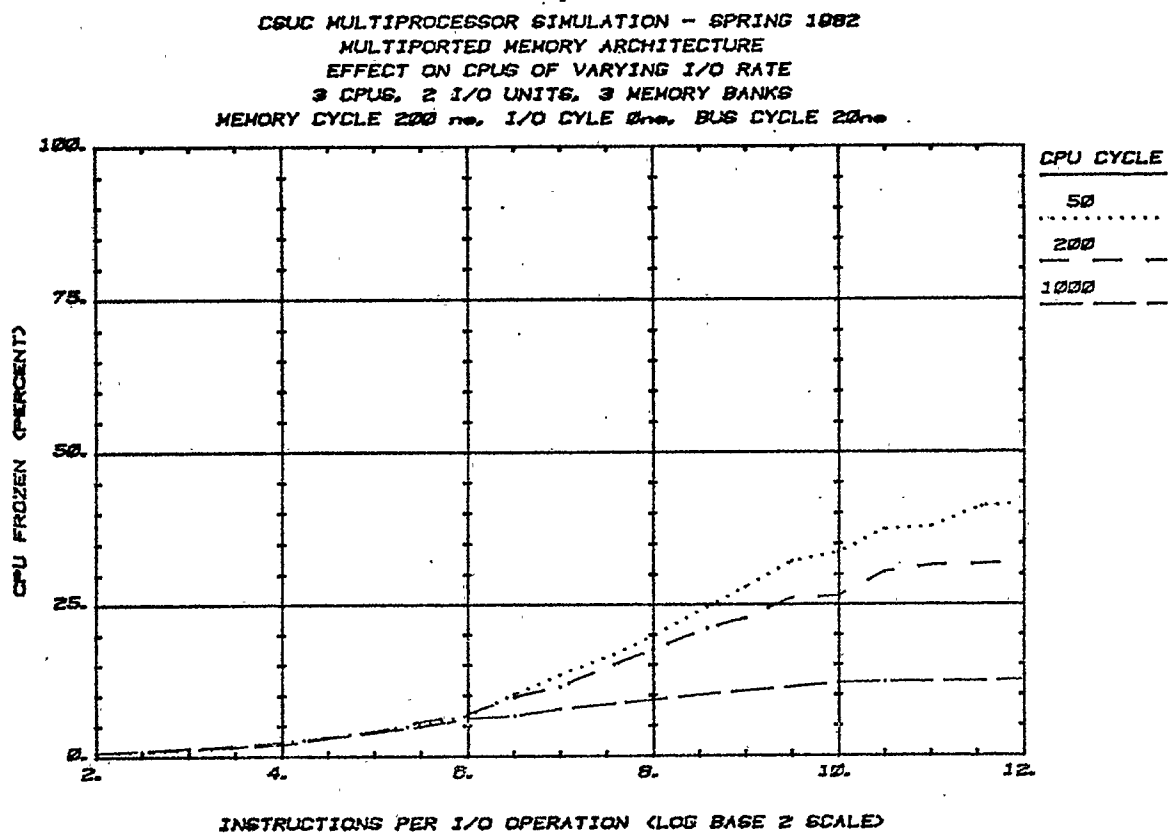


Figure 7a

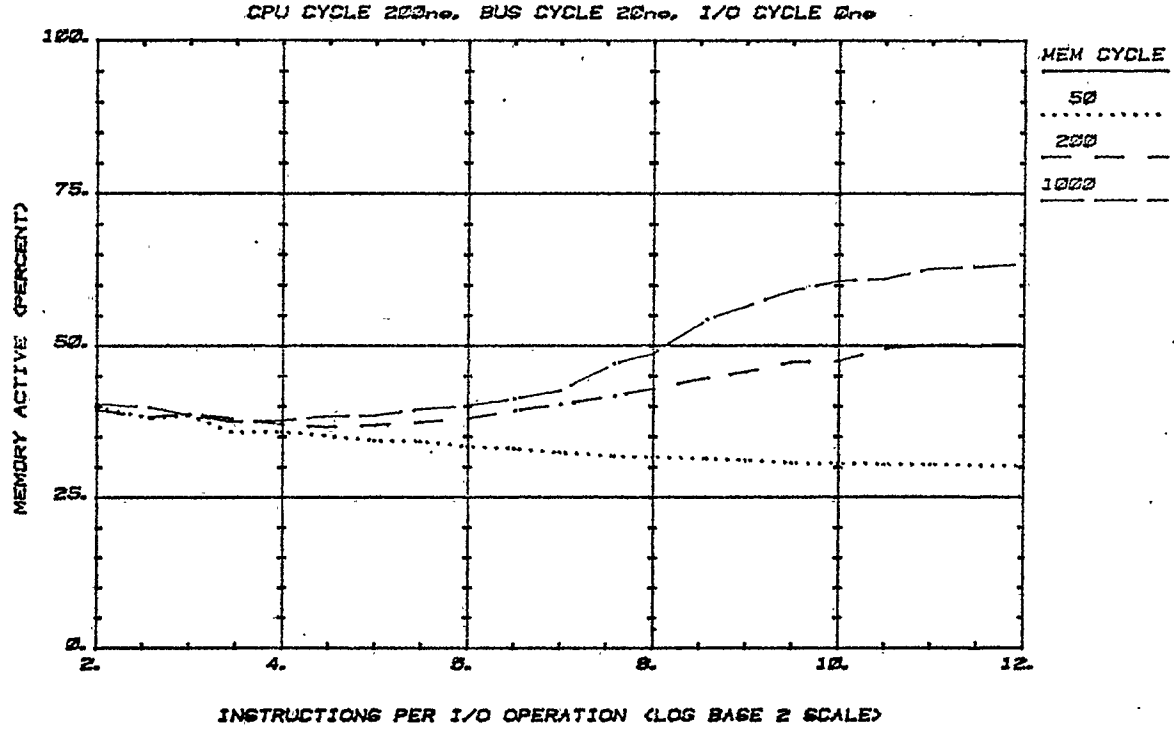
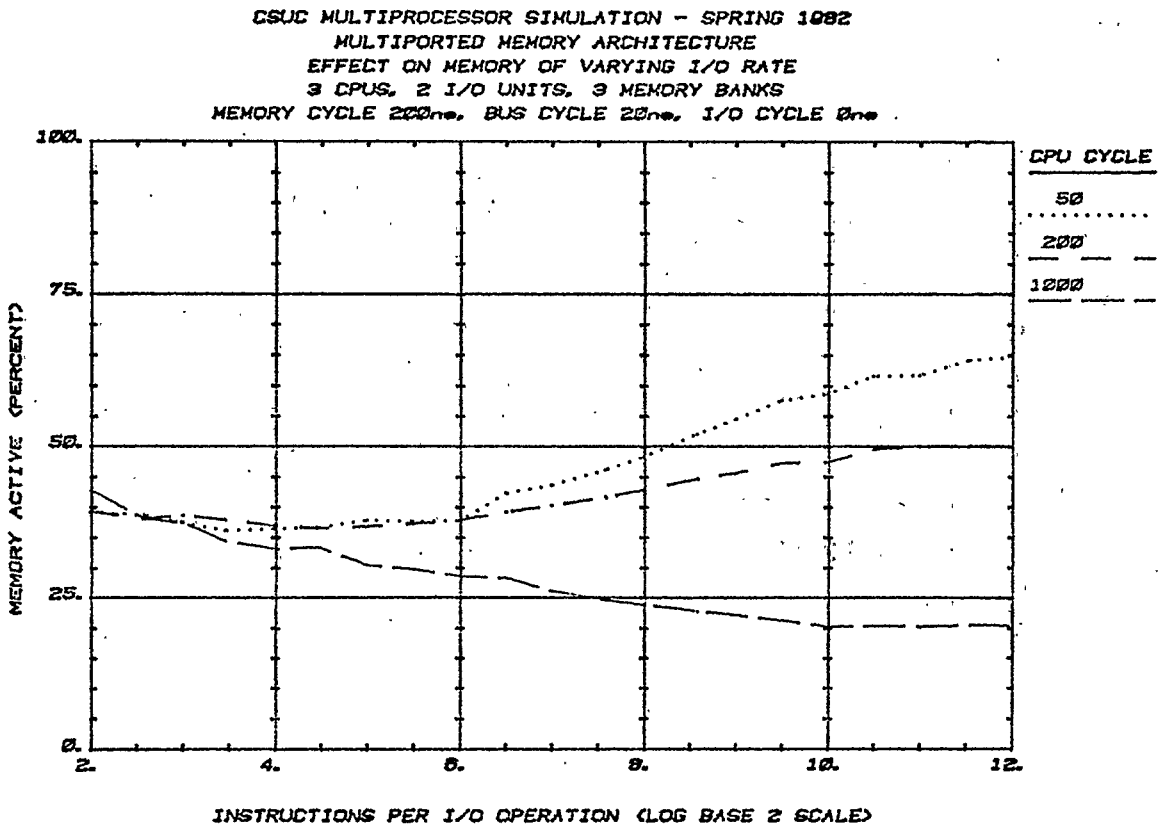


Figure 7b



CONCLUSIONS

We have attempted to compare the results of our simulation with the results published by other authors; this has not been an easy task. The main problem is that simulations of this type are necessarily complex and published accounts usually do not give sufficient information to determine how a specific model operates. To accurately compare two models, information, such as detailed activity diagrams and descriptions of allocation policies, is needed; this information is seldom included in published journal articles. In many cases, simulation results are presented but only a very brief description of the modeled system is given. For example, Bowen and Buhr (8) show how the throughput on a Common Bus system is affected by the number of processors on the system. However, they do not state what the cycle times of the CPU's and MM's are or what I/O rates are. We have found our results most useful when examining what happens to our system when a particular parameter is changed and all other variables are held constant. We look forward to comparing our results with those of other researchers when we find other published accounts with sufficient information given to make a comparison possible.

BIBLIOGRAPHY

1. Tanenbaum, A.S., Computer Networks, Prentice-Hall. Inc., New Jersey, 1981

2. Enslow, P.H., Jr., Multiprocessors & Parallel Processing, John Wiley & Sons, Inc., New York, 1974

3. Leibowitz, B.H., "Multiple Processor Minicomputer Systems", Computer Design, Oct. 1978

4. Wear, L.L, et. al., "A Comparison of Methods for Simulating Computer Bus Architectures" 1981 Winter Simulation Conference Proceedings, IEEE, 1981

5. Weitzman, C., Distributed Micro Minicomputer Systems, Prentice-Hall, Inc., New Jersey 1980

6. Uyeno, D.H. and Vaessen, W., "PASSIM: a discrete-event simulation package for Pascal" SIMULATION, Vol 35 no. 6, Dec. 1980

7. Barnett, C.C., Micro PASSIM II, Walla Walla College, College Place, Washington, 1982

8. Bowen, B.A. and Buhr, R.J.A., The Logical Design of Multiple-Microprocessor Systems, Prentice-Hall, Inc., New Jersey, 1980

Figure 1a
TIMESHARED BUS SYSTEM

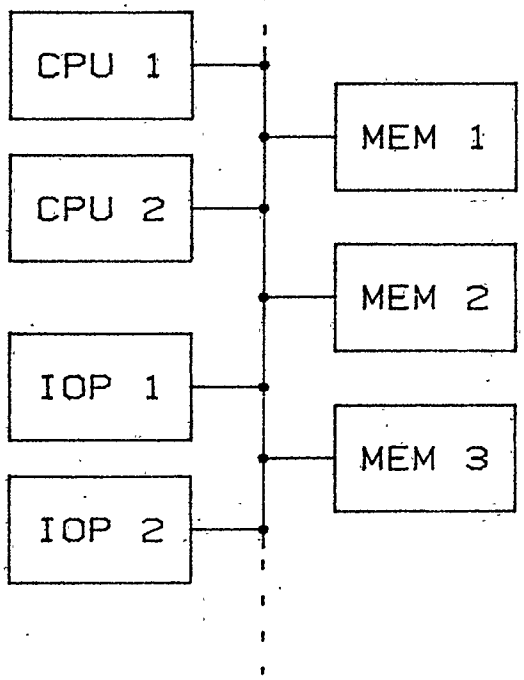


Figure 1b
MULTIPLE TIMESHARED BUSES SYSTEM

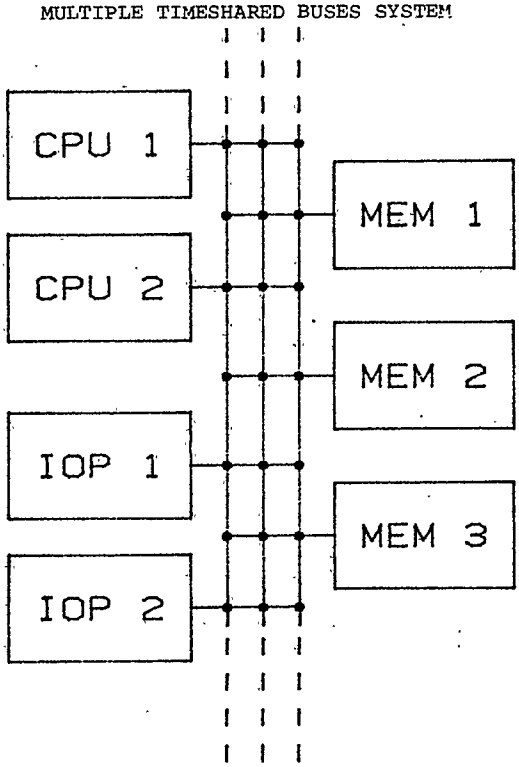


Figure 2
MULTIBUS SYSTEM

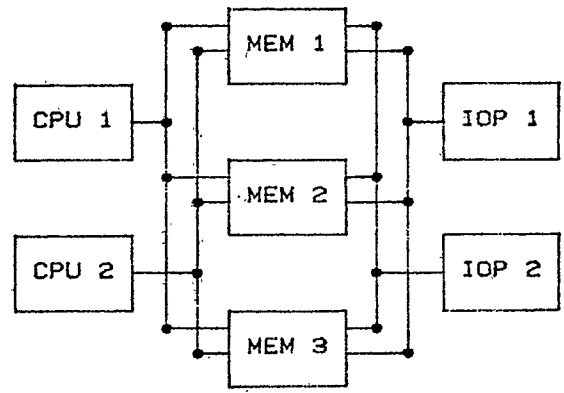


Figure 3
CROSS BAR SWITCH SYSTEM