A Study of the Multicache-Consistency Problem in Multi-Processor Computer Systems

Tarek K. Abdel-Hamid
Stuart E. Madnick

Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Approved for public release; distribution unlimited

Bus, Cache, Multicache-consistency, Multiprocessors, Performance Evaluation, Simulation

This paper is a report on an ongoing research project at the M.I.T. Sloan School of Management to study the multicache-consistency problem in multi-processor computer systems.

The nature of the consistency problem in multicache memory systems is briefly discussed, together with an explanation of the three common approaches proposed in the literature to handle it. A new solution to the problem, called the "Common-Cache / Pended Transaction Bus" (CC/PTB) - continued on reverse side.
20. Abstract

An approach, is developed and discussed. The "CC/PTB" approach attempts to minimize performance degradation by eliminating the overhead of maintaining cache-consistency. Its two distinctive features are:

Firstly, the conventional private cache per processor organization is replaced by one where a pool of cache-modules is commonly shared by all processors. Secondly, the Pended Transaction Bus (PTB) is used as the interconnection protocol that connects the processors and the cache-modules.

The performance of the CC/PTB approach is evaluated using a highly detailed simulation model with favorable results.
ABSTRACT

This paper is a report on an ongoing research project at the M.I.T. Sloan School of Management to study the multicache-consistency problem in multi-processor computer systems.

The nature of the consistency problem in multicach memory systems is briefly discussed, together with an explanation of the three common approaches proposed in the literature to handle it. A new solution to the problem, called the "Common-Cache / Pended Transaction Bus" (CC/PTB) approach, is developed and discussed. The "CC/PTB" approach attempts to minimize performance degradation by eliminating the overhead of maintaining cache-consistency. Its two distinctive features are: Firstly, the conventional private cache per processor organization is replaced by one where a pool of cache-modules is commonly shared by all processors. Secondly, the Pended Transaction Bus (PTB) is used as the interconnection protocol that connects the processors and the cache-modules.

The performance of the CC/PTB approach is evaluated using a highly detailed simulation model with favorable results.
TABLE OF CONTENTS

I. THE MULTICACHE-CONSISTENCY PROBLEM: AN INTRODUCTION ....1

II. INFOPLEX: A MULTI-PROCESSOR COMPUTER SYSTEM .............7

III. THREE COMMON APPROACHES FOR SOLVING THE CACHE-CONSISTENCY PROBLEM ...............................................11
   III.1 The "Broadcasting" Approach ....................................13
   III.2 The "Store-Controller" Approach ................................14
   III.3 The "Multics" Approach .......................................19
   III.4 Conclusion ......................................................23

IV. THE "COMMON-CACHE/PENDED TRANSACTION BUS" APPROACH ....25
   IV.1 Introduction ......................................................25
   IV.2 The Pended Transaction Bus (PTB) .............................29
   IV.3 An Application: The INFOPLEX Storage Hierarchy ..31

V. EVALUATING THE PERFORMANCE OF THE "COMMON-CACHE/PENDED TRANSACTION BUS" APPROACH .............38
   V.1 Introduction ......................................................38
   V.2 The Evaluation Tool .............................................39
   V.3 The Simulation Experiment .....................................42
   V.4 The Hardware Parameters ......................................44
   V.5 Analysis of the Simulation Results ............................46
   V.6 Conclusion ......................................................56

VI. CONCLUSION ....................................................................59

BIBLIOGRAPHY ....................................................................61

APPENDIX (I) : The "OPT" Program ......................................62
APPENDIX (II) : The "STCR" Program ....................................81
APPENDIX (III): The "CC/PTB/C" Program ................................104
APPENDIX (IV) : The "CC/PTB/P" Program ............................123
A "consistency problem" refers, in general, to a situation where two or more entries representing the same "fact" in a data base differ (i.e., are inconsistent). This, of course, can only occur when redundancy exists. In this paper we will be concerned with the "consistency problem" that arises in cache-based memory systems.

A cache memory system (Kaplan and Winder, 1973) represents a type of memory hierarchy that attempts to bridge the CPU-main memory speed gap by the use of a small, high speed random access memory whose cost per bit is higher than that of main memory, but whose total cost is relatively small because of the small size. Conceptually, this configuration has analogies with paging systems (Matick, 1977). The implementations, however, are far apart because of speed considerations. In contrast to a paging system, a cache is managed by hardware algorithms, provides a smaller ratio of memory access times (e.g., 10:1 rather than 1000:1), and deals with smaller
blocks of data (64 bytes for example rather than 4096).

In a cache system, all data are referenced by their main memory address. At any given time, a certain subset of the contents of main memory is contained in the cache level. If a processor then requests a data item in this subset, the request is serviced at the cache level.

A cache-based system works "well" for two basic reasons: First, executing programs tend to re-use instructions and data; and second, programs tend to use instructions and data near recently used instructions and data. The first property means that once information is fetched from main memory to cache, subsequent accesses to it are at cache speed. The second property means that if a request to main memory is satisfied by bringing into a cache a block of information larger than is immediately needed, the additional information is likely to be needed soon, and its presence in the cache will save one or more references to main memory.

In this paper we will refer to the block of information that constitutes the minimum amount of data which may be transmitted between the cache and main memory and which is also the allocation unit in the cache as the "cache line." All bytes of a cache line are, therefore, simultaneously all present or all absent from the cache. A directory is usually used to record the main memory addresses of all lines in the cache.

It is easy to demonstrate how inconsistency can develop in
cache-based systems due to the existence of redundancy. Consider first the simple case of a single-cache organization (Figure 1(a)).

The CPU can only access words that are in the cache. If a word that is needed for processing is not already in the cache, it will first have to be transferred from main memory to the cache. Once the word is in the cache it becomes accessible by the CPU and processing can take place. If the CPU then updates (i.e., modifies) the word, inconsistency between the copy in the cache and the copy in main memory could develop. This depends on the store algorithm used.

If a **store through** algorithm (in which the cache and main memory are updated simultaneously) is used, inconsistency will not arise. The price paid for that is a decrease in processing speed as store operations become limited by the speed of main memory. When this price is too high, **store-behind** or **store-replacement** algorithms may be used. In both cases main memory is not updated immediately, and as a result inconsistency arises. The modified word in the cache will, for "some" interval of time, be different from its unmodified version in main memory.

The more interesting case, however, is that of multicache systems. Consider the two-cache organization of Figure 1(b). What is important to emphasize here, is that the **store-through** algorithm is no longer sufficient to avoid inconsistency. Assume, for example, a word whose main memory address is \(A\), and whose current value is \(V\) is present in both cache1 and cache2. CPU1 then
(a) Single-Cache Organization

(b) Two-Cache Organization

Figure (1)
modifies the value of the word in its cache to (V.), and assuming a store-through algorithm is used, main memory is simultaneously updated. However, cache2 continues to have the unmodified version (V). If, before this inconsistency is resolved by either updating, replacing, or invalidating cache2's copy, CPU2 attempts to access the word (A), it will get what is now an invalid value (V) from its cache2.

It is time now to adopt what we believe is a more useful definition of what a "consistency problem" is. We claim that inconsistency per se is not necessarily a problem. Reconsider the case of a single-cache organization. We have already explained how inconsistency can arise for "some" interval of time when the store through algorithm is not used. During that interval of time the cache will contain the modified version of the word, while main memory will not. If, during this interval, the CPU needs to re-access the word for processing, what will happen? It will check the cache, find the word in it, and, therefore, access it i.e., no transfer from main memory will be needed. Thus, although inconsistency exists, no problem arises because the CPU will always access the updated version of the word from the cache.

With this in mind, we now adopt the following definition of a "consistency problem" (Censier and Feautrier, 1978):

"A consistency problem exists in a cache-based system if the value accessed by a CPU is not the value given by the latest store operation (by any CPU) to the same address."
It is obvious, from the above discussion, that there will be no consistency problem for single-cache organizations. These, unfortunately, are not very attractive for high performance systems.

In the next part of this paper we present a brief discussion of INFOPLEX, a highly parallel multi-processor computer system that utilizes a multicache organization. In such an organization the consistency problem is a significant one. The INFOPLEX organization will constitute the context within which we shall evaluate the different approaches for handling the multicache-consistency problem.
A research project is currently underway at the MIT Sloan School of Management aimed at investigating the architecture of a new database computer, called INFOPLEX, which is particularly suitable for large-scale information management (Madnick, 1979). The specific objectives of the project include providing substantial performance improvements over conventional architectures, supporting very large complex data bases, and providing extremely high reliability.

To provide a high performance, highly reliable, and large capacity storage system, INFOPLEX makes use of an automatically managed memory hierarchy. It is this aspect of the project that will be of relevance in our present discussion.

As a simplistic illustration, we show in Figure 2 three levels (only) of the memory hierarchy. As can be seen, this is a multicache organization. The proposed number of caches \( m \) is relatively large compared to present day systems (e.g., \( m = 32 \)). For the INFOPLEX
SLC : Storage Level Controller
MRP : Memory Request Processor

Figure (2)
objectives such a large number of caches is essential. It will, for example, help attain the high performance improvements sought (up to a 1000 fold increase in throughput over conventional architectures). In addition, it allows for the implementation of such features as dynamic reconfiguration and automatic recovery which are aimed at improving the reliability of the system.

Two important "boxes" in the design of Figure 2 are the Storage Level Controller (SLC) and the Memory Request Processor (MRP). The function of the SLC is to couple the local bus of a storage level to the global bus that connects all storage levels. In essence, the SLC serves as a gateway between levels. For example, the SLC of level (1) accepts requests to the lower storage levels from the caches and forwards them to the SLC of level (2). When the responses to these requests are ready, the level (1) SLC accepts them and sends them back to the appropriate caches.

The Memory Request Processor (MRP) performs such functions as: implementing the storage management algorithms (e.g., directing the transfer of information across a storage level); handling all the communication protocols that are peculiar to the particular storage modules (devices) at a storage level; and mapping virtual addresses into their real equivalents. (Note that an MRP is not needed at the cache level.)

The INFOPLEX organization described (briefly) above will constitute the context within which we shall study the different approaches for handling the multicache-consistency problem. For
further information on INFOPLEX the interested reader can consult the following references: (Madnick, 1975; Madnick, 1979; Lam, 1979; Lam and Madnick, 1975; and Hsu, 1980).
III. THREE COMMON APPROACHES FOR SOLVING THE
CACHE-CONSISTENCY PROBLEM

In Part (I) we explained how a consistency problem could arise in
multicache memory systems. We saw, for example, that in the two-cache
organization of Figure 1(b) a word (A) that existed in both cache1 and
cache2 could be modified to (V.) in cache1 and in main memory but not
in cache2, and thus giving rise to an inconsistent state. This
example, although rather simple, is quite adequate to demonstrate the
motivations behind the basic strategies that have been used to handle
the consistency problem in multicache systems. There are two such
strategies. First, we could restrict the "encacheability" of data
items, such that only those data items that cannot cause
inconsistencies are allowed to move into the cache level. For example,
words that can only be READ would be encacheable. On the other hand,
all data items that could potentially cause inconsistencies are
prohibited from moving into the cache level, and thus all accesses to
them are done through main memory. Word (A) of the above example
would, therefore, fall in this category, and so it (under this
strategy) would have been prohibited from moving into either cache1 or cache2. Thus accesses to word (A) by both CPU1 and CPU2 would have been made to its single copy in main memory, and no inconsistency would have resulted. The price paid, however, is that accesses to word (A) are now done at main memory speed and not at the faster cache speed.

The second basic strategy that has been employed doesn't put any such restrictions on moving data items into the cache level. The idea here is to "invalidate" a cache line when there is a risk that its contents have been modified elsewhere in the system. When a cache line is invalidated (by setting, for example, a flag in the cache directory) it is considered not in the cache. Referring again to the above example, when the value of word (A) is modified in cache1 (and in main memory) to (V.), word (A) in cache2 is invalidated. Thus if CPU2 happens to request word (A) at a later time, it will have to access it from main memory since the invalidated version (V) in its cache is considered not to exist. CPU2 will, therefore, access the valid value (V).

In the remainder of this section we will present more specific approaches to handle the consistency problem in cache-based systems. In particular, three approaches that are proposed in the literature will be discussed, namely, the "Broadcasting," the "Store-Controller," and the "Multics" approaches. The "Broadcasting" and "Store-Controller" approaches are based on the second strategy discussed above. They, though, implement it differently. The "Multics" approach, on the other hand, is based on the first strategy.
III.1. The "Broadcasting" Approach

The idea here (as mentioned in the second strategy above) is to invalidate a cache line when its contents is modified in another cache in the system. When a cache line is modified its address is broadcasted throughout the system so that other caches sharing the line would invalidate their now outdated version of it.

Every cache is connected to an auxiliary data path over which all other caches send the addresses of lines to be modified. Each cache constantly monitors this path and executes a searching algorithm on all addresses thus received. In case of a "hit," the affected line is invalidated.

When a CPU needs to read (or write) a word that doesn't exist in its own cache, the word will be seized from main memory. To ensure consistency main memory must always be kept "up-to-date." This (in general) can be guaranteed only if a store-through algorithm (in which the cache and main memory are updated simultaneously) is used. As was argued before, such a restriction is not without its cost: A decrease in processing speed as store operations become limited by the speed of main memory.

Another major drawback of this approach is that the invalidation data path must accommodate a very high traffic. The mean write rate for most processor architectures lies in the range between 10 and 30 percent (Censier and Feautrier, 1978), and thus if the number of processors is higher than two, the productive traffic between a cache
and its associated processor may be lower than the parasitic traffic between the cache and all other caches. This explains why this approach has been confined to systems with at most two caches (Censier and Feautrier, 1978).

III.2. The "Store-Controller" Approach:

The basic idea here is the same as in the "Broadcasting" approach i.e. to invalidate a cache line when its contents have been modified elsewhere in the system. It is in the implementation that the two approaches differ. Here, a "Store-Controller" SC (see Figure 3(a)) is used at the cache level to keep track of every line in every cache. The store-controller "knows," not only which lines are in which cache, but also which caches share any single line. When, therefore, a line that is shared by two or more caches is updated (i.e., modified) in one of them we do not now need to broadcast invalidation requests to all caches. We, instead, use the information in the store-controller to send invalidation requests to only those caches that are sharing the updated line, if any. In other words, the motivation behind using the store-controller is to filter out all unnecessary invalidation requests.

We will present, in a flowcharted form, an example implementation of this approach which is largely based on Tang's proposals (Tang, 1976). In this implementation, if a processor wants to write and the line is not found in its cache, then the line is always brought to the
cache so that the processor can always write to cache. The store algorithm used is the "store-replacement" algorithm. This means that when a line is modified in a cache, main memory is not concurrently updated. It is updated later when the line has to be replaced in the cache or when other caches need to have the line.

Figure 3(a) shows how the store-controller fits conceptually into the cache organization. Figures 3(b) and 3(c) show possible layouts for the directories of both the cache and the store-controller. The "status" column of the cache directory shown in Figure 3(b) needs some explanation. The status of a line can be one of three things:

1. **Private**: For a line which has been modified (with respect to main memory) or is going to be modified. A private line exists in only one cache.
2. **Non-Private**: For a line that exists in one or more caches and which has not been modified with respect to main memory.
3. **Invalid**: For a line that has been modified elsewhere and thus becomes outdated. An invalid line is considered "not in the cache."

In Figures 4 and 5 the READ and WRITE operations are illustrated respectively.

The two major drawbacks of implementing this approach in a highly parallel multi-processor computer system such as INFOPLEX where the number of caches is relatively large are:
Figure (3)
A READ instruction to CPU

Line in Cache

Check

Line not in Cache

Cache Dirac.

READ it

Another Cache Contains Line as a Private Line

STCR Checks its Direc.

Change Status of line in Cache Direc. from Private to Non-Private

Unset "Modified Flag" in Central Directory.

Transfer Line to Main Memory

Transfer Line from Main Memory to Cache

Update Directories

READ Line

END

Figure (4)
A WRITE inst. to CPU

Line in Cache

Check Cache Direct.

Line not in Cache

Non-Private

Private?

Yes

As a Private Line

Check Cent. Direc.

One or more Caches contain Line as a Non-Private Line

Update Directories

Invalidate Line in all these Caches

Line is not in other Caches

Invalidate Line in Cache

Update Directories

Other Caches Contain Line

Invalid Line in other Caches

Update Central Direc.

WRITE to Cache

END

Write to Cache

END

Figure 5
1. The size of the central directory becomes too large, which means increased time in processing it and increased costs in building it.

2. The store-controller could become a bottleneck in the system as the traffic between itself and the caches becomes very large.

III.3. The "MULTICS" Approach

This approach, which is used in Honeywell's Multics computer system (Greenberg, 1978), has two important features. Firstly, the Multics cache is a "store through" cache. This means that the cache and main memory are updated simultaneously. The second feature is that the system address space is divided into segments, each of which has associated with it names and per-user access rights which govern the ability of each potential user of the segment to read and/or write its contents (i.e., words).

Every segment is known by the system to be either "writable" or "non-writable." The non-writable class of segments, that is those to which no users have write access, is an important and statistically significant one in MULTICS. All procedures, including all parts of the operating system, utilities, libraries, translators, and so forth, fall into this category. These segments are "encacheable" by all processors. This means that their contents (or words) are allowed to "migrate" to any or all caches. Notice that when such words find their way into a cache (or more than one cache) there is no possibility for a
consistency problem to arise (for such words) since no processor can write or modify them.

For the other class of segments, those which are writable, the situation is slightly more complicated. For a segment falling in this category, there are three possible states:

1. One or more processes (users) are accessing the segment but none of them has a write access to it. The segment is encacheable by all processors but no consistency problem will arise.
2. One or more processes are accessing the segment, with at least one of them having write access. The segment becomes non-encacheable i.e., its words cannot migrate to any of the caches. The consistency problem will not arise here also since there will only be one copy (i.e., the one in main memory).
3. Only one process that has write access is accessing the segment. The segment is encacheable only to that process. And it is only in this third state that the consistency problem could possibly arise. Consider the following scenario:

Assume a certain segment S1 is addressable by only one process PROC1 which is currently running for the first time on processor CPU1. Assume also that PROC1 has write access to S1. Thus S1 is encacheable by CPU1. As long as CPU1 runs PROC1, words of S1 may be drawn into CPU1's cache and be modified by CPU1 with no problem. During this period, no other processor can address S1, for by assumption it is addressable only by PROC1, which is uniquely associated with
CPU1 during the interval in question. Thus, it is impossible for other processors to draw words of S1 into their caches as long as PROC1 is associated with CPU1. Until CPU1 leaves PROC1, there is thus no danger of words of S1 in CPU1's cache becoming outdated, as no other processor can address S1. Similarly, there cannot be words of S1 in any other processor's cache, for by assumption, CPU1 was the first and only processor to run PROC1. Thus, there is no danger that modifications to words of S1 made by CPU1 can invalidate copies in other processors' caches, since such copies cannot exist. Potential difficulty arises when CPU1 has left PROC1, and some other processor attempts to run PROC1. The first time this happens, there is no problem. Since all words in main memory are accurate, by virtue of the store-through cache, another processor, say CPU2, cannot have inaccurate data, or main memory is accurate, and we have just shown how CPU2's cache may not contain inaccurate data. However, while PROC1 runs on CPU2, CPU2 may modify words of S1 in its own cache and in main memory. Still there is no problem. Main memory is accurate, as is CPU2's cache. This can go on like this as long as processors which have never run PROC1 (since PROC1 started running) run it. However, the first time some processor which has already run PROC1 since then attempts to run it again, the scheme appears to break down. Assume CPU1 attempts to run PROC1 for the second time. There may be words of S1 in CPU1's cache from the previous time CPU1 ran PROC1. Some of these words may have been modified by PROC1 while it ran on CPU2. Thus, these words are accurate in main
memory and in CPU2's cache, but are inaccurate in CPU1's cache, for CPU2 had no way of knowing or acting upon the fact that they were in CPU1's cache.

The MULTICS solution to "its" consistency problem is simple: clear the cache of a processor upon entering a process if it was not the last processor to run that process. This is performed by the MULTICS process dispatcher, with a special processor instruction that accomplishes this task. This ensures that no words of any per-process writable segment will be found in a processor's cache if there was any possibility that any of those segments may have been modified by other processors. The operating system maintains in the control block describing each process the identity of the last processor to have run this process thus this check is easy to make when a processor is dispatched into a process.

From an INFOPLEX-type-system viewpoint, there are three major drawbacks to the MULTICS approach, all of which are performance related:

1. The class of segments that are non-encacheable can be of significant size, and thus dampening the performance gains sought by the cache organization. Note that in MULTICS there is the significant class of segments which we termed "non-writable" and which are encacheable. In data base computers, like INFOPLEX, this category which contains things like utilities, libraries, and translators will probably be much smaller, and thus decreasing the portion of encacheable segments. (There could, however, be
special applications where this doesn't apply e.g., READ-only data
base applications.) In addition, the MULTICS approach doesn't
discriminate between two processes who although both have write
access to a segment, one actually exercises the "right" and
modifies the segment, while the other doesn't. In both cases the
segment will become non-encacheable if there are other processes
that are also reading it. This means that in both cases the
system's speed will be slowed down to the speed of main memory.
2. Using a store-through algorithm has its own performance
disadvantages. As was stated earlier, it limits the speed of
store operations to that of main memory, and thus defeating the
very purpose of using a cache.
3. The procedure of clearing up the cache is also a wasteful one.
Note that a cache is always cleared if its processor wasn't the
last one to run the process. All access requests to the cleared
cache contents that would have otherwise been serviced by the
cache, must now wait for transfers from main memory. This will,
obviously, slow down the system.

III.4. Conclusion

We have analyzed the three common approaches that have been
proposed in the literature to handle the consistency problem in
multicache systems. We have considered each approach in the context of
the INFOPLEX framework presented earlier. Within this context we were
able to identify some major drawbacks in each of the approaches.
In the next section we propose a new approach to handle the cache-consistency problem in multi-processor architectures. In Part (V) we will evaluate the performance of this proposed approach.
IV. THE "COMMON-CACHE / PENDED TRANSACTION BUS" APPROACH

IV.1 Introduction

We argued in the beginning of Part (I) that in a single-cache memory system (Figure 6(a)), where there is only one access path between each level, no cache-consistency problem will arise. Once two or more caches are used, however, the potential for the problem develops.

The "traditional" approach in employing caches in multi-processor systems has been to basically replicate the structure of Figure 6(a) for each of the processors, as shown in Figure 6(b), and then solve any problems that arise. A problem that arises, of course, is the cache-consistency problem, and the three basic approaches that have been developed to handle it are those of Part (III).

Implementing any of the three approaches will obviously constitute some processing overhead. As an example, consider the
Figure (6)
"Store-Controller" approach and refer in particular to the flow-chart of Figure 5. When a CPU needs to modify a line that exists in its cache, and the line happens to be a "non-private" line, then the status of the line is first changed to "private" and the "modified flag" is set. Next, the Store-Controller's directory is checked, and if the line is found not to be shared by other caches it is modified. If, however, it happens to be shared, then messages are sent to the appropriate caches to invalidate the line, the central directory is updated, and finally the line is modified. How much overhead did we incur to maintain consistency? Well, compare the above steps with those needed for a uniprocessor system where the cache-consistency problem does not arise. In such a system, when the CPU needs to modify a line that exists in its cache, it simply proceeds and modifies it. None of the above checks, updates and messages are needed.

The concern over the processing overhead needed to maintain consistency is a legitimate one. Such overhead does undoubtedly dampen the performance gains (e.g., system throughput) which are sought by introducing caches in the first place. Attempting to minimize this overhead, therefore, seems an attractive direction for research work.

In this research endeavor, we are basically proposing an architecture that eliminates the processing overhead associated with handling the multicache-consistency problem. The architecture is depicted in Figure 6(c). The important distinction between this organization and that of Figure 6(b) is that, here, each of the cache modules is accessed by, and thus services, all of the processors. Thus, just as main memory is common to and shared by all processors,
the cache level in our proposed architecture is also common to and shared by all processors. In such a scheme there is no need to store more than a single copy of any data item at the cache level. This eliminates redundancy. And with no redundancy at the cache level, no inconsistencies can obviously arise, which in turn eliminates the need for mechanisms to maintain cache-consistency and the overhead associated with such mechanisms.

It is important to note that this architecture preserves the basic intent behind cache-based systems, namely, using a high speed memory level between the CPU and main memory in order to bridge the speed gap between the two. It, however, introduces the concern as to whether the CPU/cache bus (see Figure 6(c)) can handle the needed high volume of traffic. Comparing the Private Cache (PC) architecture of Figure 6(b) against the Common Cache (CC) architecture of 6(c), we note that the CPU/cache bus in PC handles the transaction load generated by a single processor whereas in CC it handles the load generated by all the CPUs. We can, therefore, expect that the load on the CPU/cache bus in our proposed (CC) architecture to be close to N-times that of the PC architecture, where N is the number of processors. Of course, the load will be somewhat less than exactly N-times because in PC some overhead traffic will be generated by the mechanism used to handle the cache-consistency problem, and which will not be needed in (CC).

Thus, to re-state, the Common Cache approach eliminates the cache-consistency problem (i.e., by eliminating private caches) but there is a fundamental concern that the CPU/cache bus will develop into a bottleneck that degrades the system's performance. In the next
section we introduce the Pended Transaction Bus Protocol, which we believe provides the basis for a viable solution to the problem.

IV.2 The Pended Transaction Bus (PTB)

The degree of bus utilization of a processor is a function of the physical characteristics of the bus, and the protocol used on it. The physical characteristics of the bus which include the length, voltage levels, impedance, termination, capacitance, noise immunity, and overall reflection characteristics, affect its operating speed. Ultimately, any bus is limited by the speed of electricity along a conductor (0.6 to 0.9 nanoseconds per foot typically, depending on wire characteristics). Careful electrical analysis and physical layout can optimize these parameters to achieve reasonable electrical speed.

Given an electrical bandwidth of the bus, as formed by the bus wires and the driving logic, the actual data bandwidth becomes a function of the protocol used on it. The traditionally high bus utilizations of multi-microprocessor architectures that employ the single bus as their interconnection scheme is primarily a result of the bus protocol used, and which is called the "master-slave" protocol. In such a protocol, the CPU (master) asserts a request on the bus, and the memory (slave) that receives it does the appropriate action (e.g., a READ), and then responds (with the requested data). The bus is viewed as "busy" during this entire time. A large portion of this time is actually spent waiting for the slave to complete the requested action.
The electrical and logical time to transmit the actual request and return the reply are a relatively small portion of the total bus usage time. The period of time between the request and the acknowledge is a wait interval, and no useful work is done with the bus during this time. Bus utilization could be significantly reduced if we released the bus during the wait interval. To do this we split the transaction into two parts, a request part and a reply part. The master requests the bus, and upon being granted it, sends the request to the slave at maximum speed. The slave acknowledges reception of the request, and starts to work on it. The processor then releases the bus and waits for the results. When the slave completes its task, it asks for the bus, and upon receiving it sends the reply back to the originating master at maximum speed. The time between the request and reply can be used by other masters and slaves to transfer other messages over the bus. Because the slave stores the incompletely request from its master, it is called a pended transaction, and the bus protocol, developed at M.I.T., is called a Pended Transaction Bus (PTB) protocol (Toong, et al, 1980).

In the above discussions, it was presumed that the slaves were always able to accept the master requests when presented. This would imply that each master was using different slaves to guarantee such separation. Given the shared nature of the cache data, it is likely that two (or more) masters may make requests to the same cache-memory slave. The simplest scheme to resolve this contention problem is for a busy slave to refuse a new request and make the requesting master retry at a later time when the slave becomes free. This, however, is wasteful of bus bandwidth, since the time spent to send a request out
the first time, only to be refused, is not useful. Furthermore, the slave may still be busy when the master tries again later.

Goodrich (Goodrich II, 1980) has proposed putting queues on the inputs of the slaves to buffer requests. That is, any requests that come while the slave is busy would be placed in a first-in-first-out queue, and these requests would be serviced in order when the slave is able to handle them. Such a scheme would reduce the bus load to what is actually needed for transmission, without any extra cycles. In addition, it would also improve the slave response time as seen by the master over the simple scheme described before, since the slave would have the request in its queue, and would service it as fast as it could, not just when the master is finally successful in transmitting it. Note that a queue overflow need not be fatal in this system. It can be treated like a refusal in the previous scheme, and would simply require the master to retransmit the request later. If the queue size is sufficiently large, such refusals would be rare. Finally, for best slave throughput, there should also be a queue on the output of each slave.

IV.3 An Application: The INFOPLEX Storage Hierarchy

In the above sections we have introduced an approach to the cache-consistency problem in highly parallel multi-processor computer systems, which we will call the "Common-Cache / Pended Transaction Bus" approach (CC/PTB). Its two distinctive features are: Firstly, the
conventional one to one relationship between processors and caches i.e., where each CPU has its own private cache, is replaced by an N-to-M relationship, where a pool of cache-modules is commonly shared by all processors. Secondly, we propose the use of the Pended Transaction Bus (PTB) as the interconnection scheme that connects the processors and the cache-modules.

In this section we describe how the CC/PTB architecture can be incorporated into the storage hierarchy of the INFOPLEX database computer. Schematically the proposed architecture would look like Figure 7(b). We will, henceforth, refer to this architecture as INFOPLEX/CC (for Common Cache) while referring to the original Private Cache organization (shown in Figure 7(a)) as the INFOPLEX/PC architecture.

The key INFOPLEX storage hierarchy operations are the READ and WRITE. In INFOPLEX/PC two strategies are needed to implement these operations, a strategy for the cache level and another for all other levels. The reason for this is manifested in Figure 7(a), where it can be seen that the organization of the cache level is different from the other levels of the hierarchy. In INFOPLEX/CC, on the other hand, the cache level organization is very similar to that of the lower storage levels. As a result, the strategy used to implement the READ and WRITE operations could be the same for all levels of the hierarchy with very minor provisions to account for the few differences that do still distinguish the cache level. (For example, the absence of an MRP at the cache level.)
(a) Private Cache (PC) Architecture

(b) Common Cache (CC) Architecture

**Legend:**
- **LBUS1:** Local BUS at level (1) - cache level
- **LBUS2:** Local BUS at level (2) - main memory level
- **SC:** Storage-Controller - needed only if "Storage-Controller" approach is implemented
- **SLC:** Storage Level Controller - it couples a storage level to the Global bus (i.e., serves as a gateway between levels)
- **MPP:** Memory Request Processor - performs address mapping function

*Figure (7)*
We will attempt now to explain briefly how the basic READ and WRITE operations will be implemented in the INFOPLEX/OC architecture. To a large extent this will be based on the work of Lam (Lam, 75), where a much more detailed and complete discussion is presented.

All READ and WRITE operations are performed in the highest storage level L(1) i.e., the cache level. If a referenced data item is not in L(1), it is brought up to L(1) from a lower storage level via a READ-THROUGH operation. The effect of an update to a data item in L(1) is later propagated down to the lower storage levels via a number of STORE-BEHIND operations.

When a READ request is issued by a processor, the cache level is checked to see if the requested data is in it. If the data is found in a cache-module, it is retrieved and returned to the processor. If, however, the requested data is not found in the cache level, a READ-THROUGH request is queued to be sent to the next lower level L(2) via the Storage Level Controller (SLC). As mentioned in Part (II) the SLC serves as a gateway between the storage levels of the hierarchy.

At a storage level, a READ-THROUGH request is handled by the Memory Request Processor (MRP). An MRP performs the address mapping function. It contains a directory of all the data maintained in the storage level. Using this directory, the MRP can determine if the requested data is in one of the storage devices at that level. If the data is not in the storage level, the READ-THROUGH request is queued to be sent to the next lower storage level via the Storage Level Controller (SLC).
If the data is found in a storage level L(i), the M&P maps the main memory address of the requested data item into its real address in L(i). This real address is used by the appropriate storage device to retrieve the block containing the data and then passes it to the SLC. The SLC would then broadcast the block to all upper storage levels by dividing it into fixed size packets. Each upper storage level has a buffer to receive these packets. A storage level only collects those packets that assemble into a sub-block of an appropriate size (peculiar to the storage level) that contains the requested data. This sub-block is then stored in a storage device. At L(1), the sub-block (i.e., the cache line in this case) containing the requested data is stored, and the data is finally sent to the processor that initiated the request.

In a WRITE operation, the data item is written into a cache-module, and the processor is notified of the completion of the WRITE operation. We shall assume that the data item to be written is already in L(1). (This can be realized by reading the data item into L(1) before the WRITE operation.) A STORE-BEHIND operation is next generated by the cache-module and sent to the next lower storage level. INFOPLEX uses a two-level STORE-BEHIND strategy. This strategy ensures that in a hierarchy with N levels, an updated block will not be considered for eviction from a storage level L(i), until its "parent" blocks at levels L(i+1) and L(i+2) are updated. This scheme will ensure that at least two copies of the updated data exist in the storage hierarchy at any time. The motivation behind using such a strategy is two-fold. Firstly, the reliability of the system is enhanced because at least two copies of newly written data are always maintained until the data is "securely" copied at the lowest level of
the hierarchy. Furthermore, the STORE-BEHIND strategy allows for the updating of lower storage levels to be carried out at slack periods of system operation, thus enhancing performance.

In the above discussions we didn't show how a specific cache-module can be correctly selected to handle a READ or a WRITE operation of a particular data item. In all but the cache level this function is performed by the Memory Request Processor (MRP), which maps main memory addresses into their physical equivalents at a particular level. What we need, therefore, is to augment the processors/common-cache interface to translate main memory addresses to physical addresses in the cache-modules.

There are two possible places to perform the translation operation: at the processor interface and at the cache interface. At the processor interface, the address gets translated before it reaches the bus. This requires that each processor be "informed" about all current lines at the cache level. This information would be used to translate all the main memory addresses of these lines to their equivalents at the cache level. An advantage of this is that the translation can be performed while the processor is arbitrating for the bus, and if the translation operation is fast enough, it can be done without any access time penalty.

In the second possible scheme the address gets translated at the cache-module interface. What would be needed here is a mechanism incorporated in the recognition circuitry of each cache-module that would translate the main memory address put on the bus, and that would
accordingly decide IF the desired data item is present in the cache level and if so WHERE i.e. in which cache-module and in which location in it. Both these should be done as fast as possible. Tag directory schemes incorporating set associative mapping are suggested by Mattick (Mattick, 1977).

In Part (V) we will evaluate the performance of both these schemes when implemented at the cache level of the INFOLEX storage hierarchy.
V. EVALUATING THE PERFORMANCE OF THE "COMMON-CACHE / PENDED TRANSACTION BUS" APPROACH

V.1. Introduction

To evaluate the performance of our CC/PTB approach we used the INFOPLEX multi-level storage system as our test case. Very slight modifications to the existing design were needed to incorporate "CC/PTB" into the INFOPLEX storage structure. For example, instead of using the four level memory hierarchy studied previously by Lam (Lam, 1979a) just two levels, the cache level and main memory were sufficient for our purposes.

The "CC/PTB" approach was compared against two benchmarks. Firstly, we evaluated the performance of the "Store-Controller" approach as a representative of the traditional approaches. Selecting the "Store-Controller" approach to evaluate our scheme against cannot, however, provide an effective answer to the question of how "optimal" either approach is. What is needed is an "absolute" benchmark against
which both approaches could be judged. For this purpose we evaluated the performance of a traditional private cache architecture assuming no overhead for handling the cache-consistency problem. This, then, constitutes the "performance ceiling" that no cache-consistency handling mechanism (for INFOPLEX) could possibly exceed. It is also a valuable reference point in that it tells us how close we are to an "optimal" solution.

V.2. The Evaluation Tool

The evaluation was performed by producing a simulation model in GPSS. We developed four separate GPSS programs:

1. Program "OPT" ignores the cache-consistency problem, and thus incorporates no overhead for handling it. It provides us with a ceiling on performance.
2. Program "STCR" incorporates the "Store-Controller" approach.
3. Program "CC/PTB/C" incorporates the "CC/PTB" approach with the translation operation performed at the cache-interface.
4. Program "CC/PTB/P" incorporates the "CC/PTB" approach with the translation operation performed at the processor interface.

The GPSS code for each of the above four programs is presented in a separate Appendix (Appendices I through IV). All four programs are highly detailed simulators. A widely used index that characterizes the degree of detail of a simulation model is its resolution in time, which
is defined as the shortest interval of simulated time between two consecutive events being considered (Ferrari, 1978). In our four simulations, the resolution in time is 10 nanoseconds. Such detailed simulators are likely to be more accurate and to have broader field of application than less detailed ones. However, they are certainly more expensive to design, implement, test, document, and use.

In addition to deciding on the degree of detail of the models, another basic decision had to be made concerning the workload that would drive the simulations. We decided to perform our measurements in an operating environment that is not at all uncommon in present-day computer systems, namely, running at capacity. Under such conditions, there will always be at least one transaction in the system's input queue(s) waiting to be serviced. In such a case, the performance characteristics that are of interest to us, such as throughput per unit time, become insensitive to the distribution of job arrivals.

Before concluding this section we would like to emphasize some of the structural differences between the four models, as well as some of their common characteristics. The basic structural differences are highlighted in the diagrams of Figure 8. In Figure 8(a) a two-storage-level version of the INFOPLEX storage hierarchy proposed in (Lam, 1979b) is shown. To support the "Store-Controller" approach an SC "box" (dotted in Figure 8(a)) is added to the architecture. In Figure 8(b) the architecture we proposed in Part IV to support the two versions of the "CC/PTB" approach is incorporated into the INFOPLEX storage system. Notice that in the CC/PTB architecture we deliberately maintained the same number of caches (5) as in Figure 8(a). It is
(a) Private Cache (PC) Architecture

(b) Common Cache (CC) Architecture

**Labels**

- **LBUS1**: Local BUS at level (1) - cache level
- **LBUS2**: Local BUS at level (2) - main memory level
- **SC**: Storage-Controller - needed only if "Storage-Controller" approach is implemented
- **SLC**: Storage Level Controller - it couples a storage level to the Global bus (i.e., serves as a gateway between levels)
- **MRP**: Memory Request Processor - performs address mapping function

*Figure (8)*
important to realize that while a 1:1 relationship between the CPUs and the caches is inherent in the Private Cache (PC) architectures, such is not the case for the Common Cache (CC) architectures. We, however, chose to maintain the same number of caches in both architectures (and four simulation models) to neutralize it as a factor that might affect performance.

Finally, there is a set of common characteristics that are shared by all four models, these are:

- **Degree of Multiprogramming of a CPU** = 10
- **Bus Width** = 8 bytes
- **Size of Transaction without data** = 8 bytes
- **Size of Transaction with data:**
  - at Level 1 = 8 bytes
  - at Level 2 = 64 bytes
- **Size of Data Buffers** = 10 64-byte transactions

Finally, the Pended Transaction Bus (PTB) protocol will be used in all three architectures (and four models). That is, we are committed to the PTB protocol in INFOPLEX as a result of our experimental work (at M.I.T.), which demonstrated its performance advantage.

V.3. The Simulation Experiment:

Our criterion for measuring performance in this experiment, and which will also serve as our dependent variable, will be the total system throughput as measured by the total number of transactions processed per unit of time. As for the independent variable, there
were two candidates: (1) the hit ratio, which is the percentage of
time that a referenced data item is found in the cache level; and (2)
the transaction mix i.e., the percentage of READ requests versus WRITE
requests.

The latter was chosen because we felt it is in a sense, a more
independent variable. What we mean is that the transaction mix is
largely a function of the use of the system and as such we have very
little control over it. The hit ratio, on the other hand, is
system-dependent. It can be affected by manipulating such system
parameters as the total size of the cache level and the size of the
individual cache blocks.

Thus, the hit ratio will be held constant, throughout the
experiment, at a value of 0.90 for READ requests and 1.00 for WRITE
requests. (That is, we are assuming that a WRITE of a data item is
always preceded by a READ to it.) The transaction mix i.e., the
percentage of READs, will be allowed to vary in the range from 70 % to
90 %. This is the range in which the mean READ rate lies for most
processor architectures (Censier and Feautrier, 1978).

Both the hit ratio and the transaction mix are variables that
affect the performance of the system but which are independent of the
mechanisms used to handle the cache-consistency problem. There are,
however, variables that are peculiar to the particular mechanism used,
and which influence the system's performance significantly.

In the "Store-Controller" approach the degree of sharing between
the caches is by far the most important such variable. We evaluated the "Store-Controller" approach under two operational modes, a "pessimistic" mode and an "optimistic" mode. Under the optimistic mode no sharing between caches takes place, yielding an upper bound on the performance of this approach. Under the pessimistic mode a high degree of sharing will be introduced (50 % of the cache blocks will be shared by more than one cache-module). This will then provide us with a conservative lower bound on the performance of the "Store-Controller" approach.

With respect to the "CC/PTB" approach we will also follow the above strategy, and evaluate it under both "pessimistic" and "optimistic" conditions. Under the optimistic condition we will assume the load on the cache-modules to be uniformly distributed (i.e., each of the 5 cache-modules carries 20 % of the load). And for the pessimistic case we will assume the load to be linearly distributed between 10 % at the least loaded cache-module and 30 % at the most loaded.

V.4. The Hardware Parameters

It is necessary to determine the speeds of the different hardware components in the INFOPLEX storage hierarchy. In particular, what we sought were the best possible 1985 projections for these speeds, since the first hardware prototype of the INFOPLEX data base computer was not expected before then.
The forecasts shown below constitute a realistic, but ambitious, scenario for 1985. In other words, they incorporate the fastest possible components that we envision as being available (and appropriate) for building an INFOPLEX in 1985. (Only a subset of the parameters are shown.) The bus speed \( b \) is 10 nanoseconds, the cache READ/WRITE speed \( c \) is 20 nanoseconds, and the remaining parameters are multiples of the latter, as shown. In other words, we used the cache speed as a logical building block to "build" the forecasts of the other hardware components (other than the bus). For example, if the cache READ/WRITE speed is 20 nanoseconds the READ/WRITE speed of main memory would be \( 10 \times c = 200 \) nanoseconds. The parameter values are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus speed ( b )</td>
<td>10 \text{ nanoseconds}</td>
</tr>
<tr>
<td>Cache READ/WRITE speed ( c )</td>
<td>20 &quot;</td>
</tr>
<tr>
<td>Directory Lookup ((2c))</td>
<td>40 &quot;</td>
</tr>
<tr>
<td>Directory Update ((4c))</td>
<td>80 &quot;</td>
</tr>
<tr>
<td>Storage Level Controller (SLC) speed ((2c))</td>
<td>40 &quot;</td>
</tr>
<tr>
<td>Main Memory READ/WRITE speed ((10c))</td>
<td>200 &quot;</td>
</tr>
</tbody>
</table>

Three other scenarios will be tested. The bus speed, in all three, will remain at 10 nanoseconds. The cache READ/WRITE speed, however, will take the increasing values of 40, 60, and 80 nanoseconds. And finally, the remaining parameters will maintain their relative values in terms of \( n \), the cache READ/WRITE speed. For example, when the cache READ/WRITE speed \( c \) becomes 40 nanoseconds the READ/WRITE speed of main memory will be \( 10 \times c = 400 \) nanoseconds.

Selecting several "good" 1985 scenarios reflects the fact that different options, in building an INFOPLEX, will be available to accommodate the cost/performance tradeoffs. And because the bulk of the
system's cost lies largely in the storage components, the variations in the forecasts between the different scenarios involved mainly those components.

V.5 Analysis of the Simulation Results:

As mentioned previously, our dependent variable in this experiment is system throughput. It is also the criterion we use to evaluate the performance of the cache-consistency handling mechanisms.

The throughput of any computer system is bounded by one of two factors, namely, bottlenecks in the system or the transaction load on it. In section V.2 we mentioned that our models will operate in a maximum load closed-loop environment, where new transactions are continuously generated to replace serviced ones. In such an environment, bottlenecks will definitely arise, limiting the throughput of the system.

Thus, in the process of interpreting the simulation results, we wish to identify and analyze system bottlenecks. In such an analysis, one needs to consider four major factors that directly influence the evolution of a particular system component into becoming the system's bottleneck. The four factors are:

1. Architecture: Consider for example the (a) PC and (b) CC architectures of Figure 8. In PC the local bus of level 1 (the
cache level) handles only the communications between the five caches and main memory. In the "CC/PTB" architecture the cache level bus must handle, in addition, the communications between all the processors and the cache-modules. The chances for the local bus of level 1 (LBUS1) to become the system bottleneck are, therefore, much higher in the CC architecture than they are in PC.

2. Algorithms and Protocols: The set of algorithms supported by an architecture and the protocols and mechanisms used to implement them undoubtedly influence the utilization patterns of the different architectural components. Consider for example the central role of the Store-Controller "box" in implementing the algorithms that support the READ and WRITE operations in the "Store-Controller" approach. Such a role will inevitably lead to high levels of utilization of the Store-Controller.

3. Workload: We mentioned in section V.3 that we intend to evaluate the "CC/PTB" approach using two different load distributions on the cache-modules, a uniform distribution and a linear one. In the latter case, the cache-module carrying the highest load (i.e., 30% of total load) could clearly develop into a system bottleneck.

4. Hardware Components. Characteristics: The characteristic of significance here is speed. For an example we refer again to Figure 8. All communications between level 1 and level 2 in the INFOPLEX storage hierarchy go through the Storage Level Controllers (SLCs) of both levels as well as through the Global Bus. The time needed by the Global Bus to process any of the communication messages (i.e., the time it takes to transmit the message) will usually be less than that needed by the SLC to
process the same message. This means that the SIC will always saturate before the Global Bus can develop into a bottleneck.

As part of the standard GPSS output, the utilizations of all hardware components (that are modelled as GPSS "facilities") are printed. This allows us to precisely identify the system bottleneck(s). An example is shown in Figure 9. In this case the bottleneck is LBUS1 (the local bus at level (1)) with a utilization of approximately 100%.

In Figure 10 our simulation results for the four scenarios are presented. In the discussion that follows, we will identify the different scenarios by their cache speed/bus speed ratios (\(n\)) (i.e., \(n = 2, 4, 6, \) or \(8\)) as it is a convenient parameter that completely characterizes each.

For each technology scenario (i.e., \(n\) value) seven curves are plotted. The single solid (---) curve portrays the performance of the "OPT" model, in which no mechanisms for handling the cache-consistency problem (and, therefore, no overheads) are incorporated. Thus the throughput of the "OPT" model, as is demonstrated in the figure, provides a ceiling for all the other models. The "Store-Controller" model's results are depicted by the two dash-and-dot (---) curves (S1) and (S2). Curve (S1), which is always the dominating curve, is for the case where there is no data sharing between the caches, and (S2) is when 50% data sharing is introduced. And finally, there are two curves for each of the two implementations of the "CC/PTB" approach. The two dashed (-----) curves (P1) and (P2) belong to the "CC/PTB/P"
<table>
<thead>
<tr>
<th>FACILITY</th>
<th>AVERAGE UTILIZATION</th>
<th>NUMBER ENTRIES</th>
<th>AVERAGE TIME/TRAN</th>
<th>SEIZING TRANS. NO.</th>
<th>PREEMPTING TRANS. NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB5</td>
<td>.474</td>
<td>1199</td>
<td>3.959</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBUS1</td>
<td>.999</td>
<td>6514</td>
<td>1.535</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>LBUS2</td>
<td>.702</td>
<td>1702</td>
<td>4.126</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>DRP11</td>
<td>.396</td>
<td>678</td>
<td>5.846</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRP12</td>
<td>.405</td>
<td>695</td>
<td>5.833</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>DRP13</td>
<td>.411</td>
<td>703</td>
<td>5.849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRP14</td>
<td>.369</td>
<td>629</td>
<td>5.879</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRP15</td>
<td>.398</td>
<td>681</td>
<td>5.856</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KRP1</td>
<td>.479</td>
<td>1199</td>
<td>4.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KRP2</td>
<td>.485</td>
<td>1214</td>
<td>4.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRP2</td>
<td>.297</td>
<td>719</td>
<td>3.997</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRP21</td>
<td>.198</td>
<td>496</td>
<td>4.000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure (9)
Figure (10)
implementation, while the two dotted (...) ones (C1) and (C2) are for "CC/PTB/C." The two dominating curves in both implementations, namely the (P1) and (C1) curves, are for the case where the load is uniformly distributed among all cache-modules. The two other curves (P2) and (C2) depict the performances under the linearly distributed load.

As Figure 10 demonstrates, the "CC/PTB" architecture, in both its implementations, completely dominates the "Store-Controller" in three out of the four technology scenarios. Only in the first scenario (\(n = 2\)) does the "Store-Controller" show a performance advantage and only for high READ rates. The remaining part of this section will be devoted to an analysis of the different factors affecting the performance patterns demonstrated in Figure 10. Of particular help in conducting this analysis is the information of Figure 11 depicting the system bottlenecks in all the cases tested.

There are two basic patterns that deserve separate analysis. The distinctive pattern of \(n = 2\), and the pattern common among the three other scenarios, \(n = 4, 6, \text{ and } 8\). To analyze the latter we will arbitrarily pick the case of \(n = 6\) as our analysis vehicle.

V.5.1. Case of \(n = 2\)

To many, the most surprising aspect of these results will be the almost identical shapes of the "OPT" curve, and the "Store-Controller" curve (S1). To understand why this is so, we first note from Figure 11 that in both models LBUS2 (i.e., the local bus of level 2) is the bottleneck. Now, the difference between the "Store-Controller" model
<table>
<thead>
<tr>
<th>Program</th>
<th>( n = 2 )</th>
<th>( n = 4 )</th>
<th>( n = 6 )</th>
<th>( n = 8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>% READS</td>
<td>( 70 \ 80 \ 85 \ 90 )</td>
<td>( 70 \ 80 \ 85 \ 90 )</td>
<td>( 70 \ 80 \ 85 \ 90 )</td>
<td>( 70 \ 80 \ 85 \ 90 )</td>
</tr>
</tbody>
</table>

**OPT**
- LBUS2
- SLC

**STCR**
- LBUS2
- SC

**CC/PTB/C**
- LBUS2
- LBUS1
- SLC
- \( C_1^* \)

**CC/PTB/P**
- LBUS2
- LBUS1
- SLC

*\( C_1 \) is the Cache-Module carrying 30 percent of the load*

*Figure (11)*
and the "OPT" model lies in the overhead necessary to handle the cache-consistency problem. All this overhead (in the "Store-Controller" model) is in the form of additional operations which all take place at the cache level. Thus, while removing this overhead (in the "OPT" model) will necessarily decrease the load on the cache level, it will not decrease the load on level 2, and in particular on LBUS2. Thus, since LBUS2 is the bottleneck in both cases, and since the load on it (by an "average" transaction) remains the same, the performances in both are very similar. For the same reasons, the other five models, namely, (S2), (P1), (P2), (C1), and (C2), have performances close to that of "OPT" for % READs = 80 (i.e., they all have LBUS2 as the system bottleneck).

The fact that LBUS2 is the bottleneck is itself, by the way, an interesting finding. With a hit ratio of 0.90 for READ requests and 1.00 for WRITE requests most of the "action" is clearly done at the cache level. It would, therefore, seem that LBUS1 must be saturated before LBUS2. The answer goes back to the parameter values of section V.2. Notice that the transaction size for level 2 (64 bytes) is eight times larger than that for level 1 (8 bytes). This means that a single transmission on LBUS2 will be approximately eight times longer in time than a single transmission on LBUS1. Thus, although the absolute number of transmissions on LBUS2 is less than that on LBUS1, the utilization of LBUS2 in this case is higher.

Another interesting observation relates to curve (S2) of the "Store Controller." Curve (S2) is always below curve (S1) because in (S2) the "Store Controller" (SC) is the system bottleneck with a
saturation point lower than that of LBUS2. The reason this happens is that the high degree of data sharing between the caches (and which is "orchestrated" by the Store-Controller) means a higher utilization of the Store-Controller by the "average" READ/WRITE request. Notice also that the two curves (S1) and (S2) diverge as the % of READs increases. This is merely a reflection of the pattern by which the READ and WRITE operations utilize the two respective bottlenecks, LBUS2 and SC. (Straightforward analytic calculations would demonstrate that the effective capacity of LBUS2 increases faster than does that of SC as the % of READs increases.)

Next let us turn our attention to the "OC/PTB" results. Notice first the deflections in curves (P1), (P2), (C1), and (C2). This happens because at approximately 80 % READs LBUS1 (and not LBUS2) becomes the system bottleneck in the four cases. However, even though LBUS1 is the bottleneck for both "OC/PTB/P" and "CC/PTB/C", "OC/PTB/P" clearly dominates. This simply is because an "average" READ/WRITE request utilizes LBUS1 less often in "OC/PTB/P" than it does in "CC/PTB/C." For example, in CC/PTB/P when a requested data item is found by the processor not to be in the cache level, a request is sent to main memory via LBUS1. In CC/PTB/C, on the other hand, a CPU request must first go to a cache-module (through LBUS1), only to be found unavailable, and then forwarded by the cache-module to main memory through LBUS1 again.

Notice finally the negligible effect that the load distribution on the cache-modules has on performance (i.e., curves (C1) and (C2) are similar, as well as curves (P1) and (P2)). The reasons for this are
completely analogous to the ones explaining the close resemblance between the "OPT" curve and curve (S1). In other words, changing the load distribution does not affect the utilization of LBUS1. As long as no new bottleneck develops because of the change in the load distribution, LBUS1 will remain to be the bottleneck, and thus maintain approximately the same throughput. The utilization of the cache-module that carries the highest load under the linear load distribution for both "OC/PTB/P" and "CC/PTB/C" turns out never to exceed 60 %, which is far below the 100 % mark that has to be approached before it would replace LBUS1 as the system bottleneck. This, in a sense, is very comforting to know. It shows that the behavior of the models is, to a large extent, insensitive to the shape of the load distribution on the cache-modules that we used.

V.5.2. Case of n = 6

Most of the ideas of the above discussion are applicable to the case of n=6. For example, "OPT," "OC/PTB/P".s (P1) and (P2), and "CC/PTB/C".s (C1) all have almost identical performances because they all have the same bottleneck, namely, the Storage Level Controller (SIC).

Notice, on the other hand, that because the bus is now relatively faster in comparison to the other system components, and in particular to the Store-Controller (SC), LBUS2 ceases to be the bottleneck in the two "Store-Controller" models. Instead, SC is now the bottleneck, and the degradation in performance is quite evident. However, notice that even though SC is the bottleneck for both (S1) and (S2), the
throughputs are quite different. The reason for this is that an "average" READ/WRITE request in (S2) (where there is 50% data sharing) requires more "services" from the "Store-Controller" than does an "average" READ/WRITE request in (S1). (When data sharing exists between the cache-modules, the SC has, for example, to invalidate redundant copies when a data item is modified.)

The only remaining result that deserves some explanation, is the deflection exhibited in curve (C2) of "CC/PTB/C" at % READs = 80. The reason for this behavior is that somewhere between % READs = 80 and % READs = 85 the most heavily loaded cache-module (i.e., the one carrying 30% of the load) replaces SIC as the system's bottleneck. (See Figure 11). Notice that this does not happen in "CC/PTB/P.s" curve (P2) even though the same linear load distribution is used. The reason for this is because, even though, the cache-modules in both cases are subjected to the same load distribution, they are not subjected to the same load. This, of course, is because the READ/WRITE operations in the "CC/PTB/P" implementation use the cache-modules less often.

V.6. Conclusion

The above results clearly indicate that no one approach dominates over all four technology scenarios. Technology, therefore, must remain as an element of some uncertainty.
It is important to realize, though, that what is really important in our technology forecasts is not the absolute values of the different speeds, but rather the relative values for the different hardware components. And the most important such value is the relative speed of the bus vis-a-vis the processing components that use it. Our results clearly demonstrate that the faster the bus is relative to everything else, the more appealing the "CC/PTB" approach becomes. More specifically, when the bus is four times as fast as the cache or faster (i.e., \( n \geq 4 \)), the "CC/PTB" approach provides a 20 to 60 % performance advantage over the "Store-Controller" approach.

But, what perhaps is the most interesting finding, is the fact that for three out of the four technology scenarios (with \( n \geq 4 \)) the performance of our "CC/PTB" scheme is very close to that of "OPT."

Notice that in the above statements no attempt was made to single out any of the two different implementation schemes of the "CC/PTB" approach. One of the interesting findings in the simulation results is that the performances of both schemes are very close indeed. Our own intuition was that the "CC/PTB/P" implementation would display a performance advantage. It was clear, that by utilizing "fast" processors that would overlap address translation with arbitrating for the bus, the utilizations of the bus and the cache-modules would decrease. Although the utilizations were indeed lower, this did not materialize into the higher performance we anticipated. The reason: the execution of the INFOPLEX storage hierarchy operations and storage mechanisms was such that the storage level controller (SLC), whose utilization is independent of the "CC/PTB" scheme used, would, in most
cases, be the first component to saturate. The only exception to this is when, under the linear load distribution case, the most heavily utilized cache-module developed into the system's bottleneck. In such a case the higher performance potential of the "CC/PTB/P" scheme was indeed realized.
VI. CONCLUSION

This paper is a report on an ongoing research effort at the M.I.T. Sloan School of Management to study the multicache-consistency problem in highly parallel multi-processor computer systems.

There are three basic approaches proposed in the literature to handle the multicache-consistency problem: The "Broadcasting" approach, the "Store-Controller" approach, and the "Multics" approach. However, serious drawbacks could be identified in each. A new approach called the "CC/PTB" was, therefore, developed. It attempts to minimize performance degradation by minimizing the overhead of maintaining cache-consistency.

The "CC/PTB" approach was implemented in the INFOPLEX storage hierarchy and evaluated using simulation modeling. The results are very favorable.
This work opens up many areas for further investigation. No mention was made in this paper, for example, of the replacement algorithms at the cache level. It would be interesting to find out the value of implementing a "sophisticated" algorithm such as the "Least Recently Used" (LRU) algorithm (or versions of it) as opposed to a naive algorithm e.g., random replacement that requires much less hardware overhead.

We have assumed, as is common in the literature, that all WRITE operations for a particular data item are preceded by READ operations. Relaxing this constraint, and developing efficient algorithms to exploit both this relaxation and the architecture of the storage hierarchy is definitely worth investigating.

And finally, the "CC/PB" architecture should be exploited in the development of algorithms that would improve the reliability of the data storage hierarchy. The automatic data repair algorithms, for example, are particularly interesting and promising.
BIBLIOGRAPHY


APPENDIX (I) : The "OPT" Program
FILE: OPT V51.0B A4 CONVERSATIONAL MONITOR SYSTEM PAGE 002

INITIAL $\text{MODEL,102}$ MODEL OPT
INITIAL $\text{PS,00,10}$ DEGREE OF MULTIPROG PER CPU
INITIAL $\text{PS,00,700}$ % READ REQ
INITIAL $\text{PS,00,900}$ CONDITIONAL PROB OF FINDING DATA IN L1
INITIAL $\text{PS,00,1000}$ IN L2
INITIAL $\text{PS,00,1,2}$ DEVICE SERVICE TIME IN L1
INITIAL $\text{PS,00,2,4}$ IN L2
INITIAL $\text{PS,00,1,1}$ BUS SERVICE TIME FOR 8 BYTE BLOCK
INITIAL $\text{PS,00,2,8}$ BUS SERVICE TIME FOR 64 BYTE BLOCK
INITIAL $\text{PS,00,3,1}$ BUS TIME BETWEEN CPU AND CACHE
INITIAL $\text{PS,00,4,4}$ DIRECTORY LOOK UP
INITIAL $\text{PS,00,4,4}$ CONTROLLER SERV TIME
INITIAL $\text{PS,00,3,8}$ TIME TO USE REPL ALG & STORE IN L1
INITIAL $\text{PS,VER,10000}$ SIMULATION TIME

************************************************************

* SAVEVALUES

* NTXX TOTAL TXN PROG.
* SUMX TOTAL EXEC TIMES
* SUMW TOTAL WAIT TIMES
* SUMT TOTAL ELAPSED TIME

************************************************************

* VARIABLES

************************************************************

MRSP FYVARIABLE (PS,MT/XSNXN) MEAN RESP TIME
TXNT VARIABLE P3-P2 TXN ELAPSED TIME
TXNW VARIABLE P3-P2-P4 TXN WAIT TIME
TXNX VARIABLE P4 TXN EXEC TIME

************************************************************

* TABLES

************************************************************

TXNT TABLE STXT,100,100,100
TXNW TABLE STAXX,100,100,100
TXNX TABLE STAXX,100,100,100

************************************************************

* FUNCTIONS


CONVERSATIONAL MONITOR SYSTEM

FILE: CPT /S1JOB A4

- By FOR READ-THROUGH

RTOK2 B'A'IBLE FNUSBS1+SNFSTI21

- By FOR ...

CHK1 B.A'IBLE FNUSBS1*SNF5R3K1
DK51 B.A'IBLE FNUSBS1*SNF5SOK1
DK51 B.A'IBLE FNUSBS1*SNF5SOK1
KDT11 B.A'IBLE FNUSBS1*SNFSTD11
KDT11 B.A'IBLE FNUSBS1*SNFSTD11
KDT12 B.A'IBLE FNUSBS1*SNFSTD12
KDT13 B'A'IBLE FNUSBS1*SNFSTD13
KDT14 B'A'IBLE FNUSBS1*SNFSTD14
KDT15 B'A'IBLE FNUSBS1*SNFSTD15
KDA11 B'A'IBLE FNUSBS1*SNFSAI11
KDA12 B'A'IBLE FNUSBS1*SNFSAI12
KDA13 B'A'IBLE FNUSBS1*SNFSAI13
KDA14 B'A'IBLE FNUSBS1*SNFSAI14
KDA15 B'A'IBLE FNUSBS1*SNFSAI15

- By FOR INTER LEVEL COMM

KRR12 B'A'IBLE FNUSBS1*SNF5SIK2
KKS12 B'A'IBLE FNUSBS1*SNF5SIK2
KKG12 B'A'IBLE FNUSBS1*SNF5SOK2
KRT21 B'A'IBLE FNUSBS1*SNFSTI21
KRAM1 B'A'IBLE FNUSBS1*SNFSTI1

- By FOR L/2, CPS

KRR2 B'A'IBLE FNUSBS2*SNF5RI2
KRS2 B'A'IBLE FNUSBS2*SNF5RI2
KRT2 B'A'IBLE FNUSBS2*SNF5RI2
KRA2 B'A'IBLE FNUSBS2*SNF5AI2
KRD2 B'A'IBLE FNUSBS2*SNF5AI2
RDR21 B'A'IBLE FNUSBS2*SNF5R121
RDS21 B'A'IBLE FNUSBS2*SNF5SI21
FILE: OPT VS1 JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 005

**MACROS**

**MACRO - USE**

**#A DISTANCE**

**#B USAGE TIME**

USE STARTMACRO
SEIZE #A
ADVANCE #B
ASSIGN #A,#B
RELEASE #A
ENDMACRO

**MACRO - SEND**

**#A FROM**

**#B TO**

**#C VIA**

**#D TRANSIT TIME**

**#E BV FOR SEND DP**

SEND STARTMACRO
TEST E #E,#I
ENTER #B
SEIZE #C
ADVANCE #D
ASSIGN #A,#D
RELEASE #C
LEAVE #A
ENDMACRO

**MACRO - FINI**
FINI  START MACRO
MARK  3
SAVE VALUE W'Z--
SAVE VALUE 5'W--.SSTAX
SAVE VALUE 5'W--.SSTAX
SAVE VALUE 5'W--.SSTAX
SAVE VALUE W'W--.SSTAX
ASSIGN 1.0
ASSIGN 2.0
ASSIGN 3.0
ASSIGN 4.0
END MACRO

BEGIN SIMULATION

SIMULATE

CPU #1

RMULT 3, 5, 7, 9, 11, 13, 15, 17
CPU1 GENERATE ...$WAXMP...F
STAR1 PRIORITY 9  SET HIGH P FOR NEW TXN
MARK 2  ARRIVAL TIME
ASSIGN 1.0  CPU ID
ADVANCE X$28.5
TRANSFER .XS$=EDA.WWW1.RRR1
RRR1 TRANSFER .XS$=I11.NIN11.RIN11

DATA IS IN DATA CACHE

RIN11 ENTER RIC11  PUT TXN IN READ REQ BUFFER
USE MACRO DRP11.XSREX1  SEARCH AND READ CACHE
LEAVE RIC11  FREE BUFFER
ADVANCE X$C8.5
FINI MACRO TRANSFER .STAR1  A NEW TXN

DATA IS NOT IN CACHE
FILE: OPT 051-03 A4

CONVERSATIONAL MONITOR SYSTEM

*******************************
* DATA IS NOT IN CACHE *
*******************************

MIN12 ENTER R1D12 PUT IN READ REQ BUFFER
USE MACRO D1P12,X$RERX SEARCH DIRECTORY
PRIORITY 0 RESET PRIORITY
SEND MACRO R1D12,R0K1,LS151,X$Ste1,BS$E1K1
TRANSFER .COMR TO COMMON CODE FOR READ

*******************************
* WRITE REQUEST TO CACHE *
*******************************

WIN2 ENTER S1D12 PUT TXN IN WRITE REQ BUFFER
USE MACRO D1P12,X$RDEX1 WRITE DATA IN CACHE
PRIORITY 0 RESET TXN PRIORITY
SEND MACRO S1D12,S0K1,LS151,X$Ste8,BS$E5K1
SPLIT 1,COMMON
FINI MACRO
TRANSFER .STAR2 A NEW TXN

*******************************
* CPU #3 *
*******************************

CPU3 GENERATE ...X$WAXMP,...F
STAR3 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.3 CPU ID
ADVANCE X$CB:5
TRANSFER .X$S840D,WWW3,RRR3
RRR3 TRANSFER .ASF141,NIN13,RIN13

*******************************
* DATA IS IN DATA CACHE *
*******************************

RIN13 ENTER R1D13 PUT TXN IN READ REQ BUFFER
USE MACRO D1P13,X$RDEx1 SEARCH AND READ CACHE
LEAVE R1D13 FREE BUFFER
FILE: OPT VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 009

ADVANCE X$CBUS

FINI MACRO TRANSFER ,STAR3 A NEW TXN

**********************************************************************
* DATA IS NOT IN CACHE
* **********************************************************************

N1N13 ENTER RID13 PUT IN READ REQ BUFFER
USE MACRO DAP13,X$REX SEARCH DIRECTORY
PRIORITY 0 RESET PRIORITY
SEND MACRO RID13,R0K1,LBUS1,X$BEK1,BV$OKR1
TRANSFER ,COMR TO COMMON CODE FOR READ

**********************************************************************
* WRITE REQUEST TO CACHE
* **********************************************************************

W1W13 ENTER SID13 PUT TXN IN WRITE REQ BUFFER
USE MACRO DAP13,X$REX WRITE DATA IN CACHE
PRIORITY 0 RESET TXN PRIORITY
SEND MACRO SID13,R0K1,LBUS1,X$BEK1,BV$OKR1
SPLIT 1,COM
FINI MACRO
TRANSFER ,STAR3

**********************************************************************
* CPU 44
* **********************************************************************

CPU4 GENERATE ...,X$MAXMP,...F
STAR4 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1,4 CPU ID
ADVANCE X$CBUS
TRANSFER ,X$READ,W1W13,R1R4
R1R4 TRANSFER ,X$PIN1,N1N14,R1N14

**********************************************************************
* DATA IS IN DATA CACHE
* **********************************************************************
FILE: JPT  VS1JOB  A4  CONVERSATIONAL MONITOR SYSTEM  PAGE 010

---------
RIN14 ENTER RID14  PUT Txn IN READ REQ BUFFER
USE MACRO DRPI4,X$READX  SEARCH AND READ CACHE
   .LEAVE RID14  FREE BUFFER
   ADVANCE X$CBUS
FINI MACRO
   TRANSFER ,STAR4  A NEW Txn
---------

---------
* DATA IS NOT IN CACHE *
* *
---------
MIN14 ENTER RID14  PUT IN READ REQ BUFFER
USE MACRO DRPI4,X$REX  SEARCH DIRECTORY
   PRIORITY 0  RESET Txn PRIORITY
SEND MACRO RID14,RDK1,LBUS1,X$REX,6BSMK1
   TRANSFER ,CMCR  TO COMMON CODE FOR READ
---------

---------
* WRITE REQUEST TO CACHE *
* *
---------
MIN14 ENTER S1D14  PUT Txn IN WRITE REQ BUFFER
USE MACRO DRPI4,X$READX  WRITE DATA IN CACHE
   PRIORITY 0  RESET Txn PRIORITY
SEND MACRO S1D14,SOK1,LBUS1,X$REX,6BSMK1
   SPLIT 1,CMCR
FINI MACRO
   TRANSFER ,STAR4  A NEW Txn
---------

---------
* CPJ =5 *
* *
---------
CPUS SEVERE  .X$MAPX,F
STAR5 PRIORITY 9  SET HIGH P FOR NEW Txn
MARK 2  ARIVAL TIME
ASSIGN 1.5  CPU ID
ADVANCE X$CBUS
TRANSFER .X$READ,WWW5,RRRS
RRRS TRANSFER .X$PIN1,NIN15,RIN15
FILE: OPT VS1JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 011

**********************
** DATA IS IN DATA CACHE **
**********************

RINI5 ENTER RID15 PUT TXN IN READ REQ BUFFER
USE MACRO DRP15,X$RDX1 SEARCH AND READ CACHE
LEAVE RID15 FREE BUFFER
ADVANCE ASCBUS
FINI MACRO TRANSFER .STARS A NEW TXN

**********************
** DATA IS NOT IN CACHE **
**********************

NINI5 ENTER RID15 PUT 'N READ REQ BUFFER
USE MACRO DRP15,X$REX SEARCH DIRECTORY
SEND MACRO RID5,ROK1,LBUS1,X$BEK1,BV$OKR1
TRANSFER .COMR TO COMMON CODE FOR READ

**********************
** WRITE REQUEST TO CACHE**
**********************

WWW5 ENTER SIDI5 PUT TXN IN WRITE REQ BUFFER
USE MACRO DRP15,X$RDX1 WRITE DATA IN CACHE
SEND MACRO SIDI5,SOK1,LBUS1,X$BEK8,BV$OK81
SPLIT 1,COMR
FINI MACRO TRANSFER .STARS A NEW TXN

**********************
** COMMON CODE FOR READ REQUEST **
**********************

COMR ASSIGN 11,0
FILE: DPT  VSJOB  A4  CONVERSATIONAL MONITOR SYSTEM  PAGE 012

USE MACRO  KRP1,X$KEX
SEND MACRO  ROK1,RIK2,GBUS,X$BEK1,BV$KR12
USE MACRO  KRP2,X$KEX
SEND MACRO  RIK2,RIK2,GBUS2,X$BEK1,BV$KR22
USE MACRO  RRP2,X$REX

-----------------------------------------------
* READ DATA IS FOUND IN L(2)

-----------------------------------------------
* DATA IS IN D21

-----------------------------------------------
RRR21 ASSIGN  11.0
SEND MACRO  RIR2,RID21,GBUS2,X$BEK1,BV$RDR21
USE MACRO  DRP21,X$DEX2
SEND MACRO  RID21,RDK2,GBUS2,X$BEK8,BV$DK2

-----------------------------------------------
* READ-THROUGH TO L(1)

-----------------------------------------------
USE MACRO  KRP2,X$KEX
SEND MACRO  T0K2,TIK1,GBUS,X$BEK8,BV$RT0K2

-----------------------------------------------
* STORE DATA INTO L(1) AS RESULT OF A READ-THROUGH

-----------------------------------------------
STOR1 ASSIGN  11.0
USE MACRO  KRP1,X$KEX
FILE: OPT V51JOB A4 CONVERSATIONAL MONITOR SYSTEM

SPLIT 1, FNSWICH
TERMINATE

RT STORE INTO D11

WWW1 ASSIGN 11, 0
SEND MACRO TIK1, TID11, LBUS1, X$BEX8, BV$KDT11
USE MACRO DRP11, X$RPLR
TRANSFER , X$POV1, MOV11, OVL11
NOV11 LEAVE TID11
ADVANCE X$CBUS
FINI MACRO
TRANSFER , STAR1
OVL11 SPLIT 1, OVF11
ADVANCE X$CBUS
FINI MACRO
TRANSFER , STAR1
OVL11 ASSIGN 11, 0
SEND MACRO TID11, OOK1, LBUS1, X$BEX1, BV$DKO1
TRANSFER , OVL1

RT STORE INTO D12

WWW2 ASSIGN 11, 0
SEND MACRO TIK1, TID12, LBUS1, X$BEX8, BV$KDT12
USE MACRO DRP12, X$RPLR
TRANSFER , X$POV1, MOV12, OVL12
NOV12 LEAVE TID12
ADVANCE X$CBUS
FINI MACRO
TRANSFER , STAR2
FILE: OPT
VS1JOB A4
CONVERSATIONAL MONITOR SYSTEM

SEND MACRO AOK2,AIK1,GBUS,X$BEX1,BV$KKA21
USE MACRO KRPI,X$REX
SPLIT 1,FN$SICHA
TERMINATE

* ACK HANDLED BY D11 *

AAA11 ASSIGN 11.0
SEND MACRO AIK1,AID11,LIBUS1,X$BEX1,BV$KDA11
USE MACRO DPP11,X$REX
LEAVE AID11
TERMINATE

* ACK HANDLED BY D12 *

AAA12 ASSIGN .1.0
SEND MACRO AIK1,AID12,LIBUS1,X$BEX1,BV$KDA12
USE MACRO DRP12,X$REX
LEAVE AID12
TERMINATE

* ACK HANDLED BY D13 *

AAA13 ASSIGN 11.0
SEND MACRO AIK1,AID13,LIBUS1,X$BEX1,BV$KDA13
USE MACRO DRP13,X$REX
LEAVE AID13
TERMINATE

* ACK HANDLED BY D14 *

PAGE 017
CONVERSATIONAL MONITOR SYSTEM

FILE: OPT VS1JGB A4

---------
AAA14 ASSIGN 11.0
USE MACRO DRP14.X$REX

LEAVE AID14
TERMINATE

---------
AAA15 ASSIGN 11.0
SEND MACRO AIK1.AID15.L3.S1.X$BEX1.BV$KDA15
USE MACRO DRP15.X$REX

LEAVE AID15
TERMINATE

---------
SIMULATION CONTROL

GENERATE X$TIMER
TERM INATE 1
START 1
END

$ END
APPENDIX (II): The "STCR" Program
FILE: STCR VS1JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 001

//TAR1 JOB TAR.
// PROFILE='DEFER'.
// TIME=5
// *PASSWORD SCUBA
//GPSS PROC
//C EXEC PGM=SCUBA,TIME=8,LIST= OFF
//STEP1 DD DSNAME=LIBRARY,DISP=SHR
//OUTPUT DD DSN=S2.TFILE=R1,T,DCB=BLKSIZE=931
//DINTEG DD UNIT=SCRATCH,SPACE=CYL,(1,1)),DCB=BLKSIZE=1880
//DSYMTAB DD UNIT=SCRATCH,SPACE=CYL,(1,1)),DCB=BLKSIZE=7112
//DREPTGEN DD UNIT=SCRATCH,SPACE=CYL,(1,1)),DCB=BLKSIZE=800
//DINTER WORK DD UNIT=SCRATCH,SPACE=CYL,(1,1)),DCB=BLKSIZE=2680
// PEND
//STEP1 EXEC GPSS,PARM=C,T,LIMIT=9
//DINPUT DD *
  REALLOCATE FN.5,QUE.10,FAC.50,BVR.200,BLO.2000,VAR.50
  REALLOCATE FSV.50,HSV.10,CCY.40000

*************************************************************************************

TXN PARM USAGE

* P1 CPU ID
* P2 TXN ARRIVAL TIME
* P3 TXN COMPL TIME
* P4 TXN EXEC TIME
* P11 DUMMY

*************************************************************************************

MODEL COMPONENTS

* BUSES: GBUS, LBUS1...
* CACHES: D11,...,D15
* LEVEL CONTAB: K1,K2
* RED PROCES: R1,R2
* DEVICES: D21
* STORAGE: R1, R0
* STORAGE: S1, S0
* STORAGE: T1, T0
* STORAGE: A1, A0
* STORAGE: D1, D0

*************************************************************************************

MODEL PARAMETERS

*************************************************************************************

INITIAL X$MODEL   MODEL STCR
<table>
<thead>
<tr>
<th>INITIAL</th>
<th>XS$RAAMP,10</th>
<th>DEGREE OF MULTIPROG PER CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL</td>
<td>XS$READ,900</td>
<td>% READ REQUESTS = 0</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$PIN11,900</td>
<td>CONDITIONAL PROB OF FINDING DATA IN L1</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$PIN2,1000</td>
<td>IN L2</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$DEK1,2</td>
<td>DEVICE SERVICE TIME IN L1</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$DEK2,4</td>
<td>IN L2</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$BEX1,1</td>
<td>BUS SERVICE TIME FOR 8 BYTE BLOCK</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$BEX8,8</td>
<td>BUS SERVICE TIME FOR 64 BYTE BLOCK</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$BUS,1</td>
<td>BUS SERVICE TIME BETWEEN CPU &amp; CACHE</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$REX,4</td>
<td>DIRECTORY LOOK UP TIME</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$REX,4</td>
<td>CONTROLLER SERV TIME</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$RPL,8</td>
<td>TIME TO USE RPL ALG AND STORE IN L1</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$DEK1,6</td>
<td>LOCK PLUS READ TIME OF CACHE</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$TIMER,1000</td>
<td>SIMULATION TIME</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$PRV1,1000</td>
<td>WHEN READING (NOT IN CACHE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>% OF TIME ANOTHER CACHE DOES NOT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTAIN THE LINE AS A PRIVATE LINE</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$PRV2,1000</td>
<td>WHEN WRITING A LINE (&amp; IN CACHE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROB. LINE HAS NOT JUST DECLARED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRIVATE BY ANOTHER CACHE</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$SHR,1000</td>
<td>WHEN NOT WRITING (&amp; IN CACHE &amp; LINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOT JUST DECLARED PRIVATE BY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ANOTHER CACHE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROB THAT ONE OR MORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OTHER CACHES DON'T SHARE THE LINE</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$CNT,1000</td>
<td>PROB THAT STORE-BEHIND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACK COUNTER = 0</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$CCS,4</td>
<td>STORE CONTROLLER SEARCHES DIRECT</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$DEX,8</td>
<td>WRITE INTO CACHE UPDATE DIRECT</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$CCU,8</td>
<td>STORE UPDATES ITS DIRECT</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$CSS,8</td>
<td>STORE SEARCHES &amp; UPDATES ITS DIRECT</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$SEXT,2,2</td>
<td>CACHE RECEIVES A CHANGE OF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATUS ACK ... % SO IT WRITES</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$EEX,8</td>
<td>CACHE UPDATES DIRECT</td>
</tr>
<tr>
<td>INITIAL</td>
<td>XS$DECR,4</td>
<td>CACHE DECR &quot;STORE-BEHIND ACK CNT&quot;</td>
</tr>
</tbody>
</table>

---

**Save Values**
- **NIN** TOTAL TXN PROC.
- **SUMX** TOTAL EXEC TIMES
- **SUMY** TOTAL WAIT TIMES
- **SUMT** TOTAL ELAPSED TIMES

---

**Variables**
- **MIN** VARIABLE P3-P2
- **MEAN RESP TIME** TXN ELAPSED TIME
FILE: STCR VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

TRAN VARIABLE P3-P2-P4
TRAN VARIABLE P4

----------------------

* TABLES *

----------------------

TRAN TABLE .STAN,100,100,100
TRAN TABLE .STAN4,100,100,100
TRAN TABLE .STAN,100,100,100

----------------------

* FUNCTIONS *

----------------------

WORD FUNCTION P1.05
1,WWW41/2,WWW12/3,WWW13/4,WWW14/5,WWW15

WORD FUNCTION P1.05
1,AAA41/2,AAA12/3,AAA13/4,AAA14/5,AAA15

----------------------

* STORAGE FOR L(1) *
* CACHES *

----------------------

STORAGE SSDID11,10/SSSID11,10/SSSID11,10/SSSID11,10
STORAGE SSDID12,10/SSSID12,10/SSSID12,10/SSSID12,10
STORAGE SSDID3,10/SSSID13,10/SSSID13,10/SSSID13,10
STORAGE SSDID4,10/SSSID14,10/SSSID14,10/SSSID14,10
STORAGE SSDID5,10/SSSID15,10/SSSID15,10/SSSID15,10

----------------------

* STORAGE FOR DEVICES *

----------------------

STORAGE SSID21,10/SSID21,10/SSID21,10

----------------------

* STORAGE FOR REQ PROC *

----------------------

STORAGE SSR1R1,10/SSSR1,10/SSSR1,10/SSSR1,10/SSSR1,10/SSSR1,10
STORAGE SSR1R2,10/SSSR2,10/SSSR2,10/SSSR2,10/SSSR2,10/SSSR2,10
FILE: STCR  VS1JOB  A4  CONVERSATIONAL MONITOR SYSTEM  PAGE 004

- STORAGE FOR K1
  
  ---------------
  STORAGE  $R0K1,10/$S$OK1,10/$S$IK1,10/$S$AIK1,10/$S$OK1,10

- STORAGE FOR K2,K3,K4
  
  ---------------
  STORAGE  $R1K2,10/$S$IK2,10/$S$IK2,10/$S$AIK2,10/$S$OK2,10
  STORAGE  $R0K2,10/$S$OK2,10/$S$OK2,10/$S$OK2,10/$S$OK2,10

- BOOLEAN VARIABLES
  
  ---------------

- BY FOR READ-THROUGH
  
  ---------------

D3K11 VARIABLE FNUSLBUS1*SNFSOIR1*SNFSOK1
R3OK2 VARIABLE FNUSLBUS*SNFSIK1

- BY FOR L1(1)
  
  ---------------

D3R1 VARIABLE FNUSLBUS1*SNFSRIR1
D3R51 VARIABLE FNUSLBUS1*SNFSRIR1
D3R41 VARIABLE FNUSLBUS1*SNFSAIR1
D3R01 VARIABLE FNUSLBUS1*SNFSOIR1
R3R1 VARIABLE SNFSRIR1
S3R1 VARIABLE SNFSRIR1
R3K1 VARIABLE FNUSLBUS1*SNFSRCK1
R3D51 VARIABLE FNUSLBUS*SNFSID11
R3D512 VARIABLE FNUSLBUS1*SNFSID12
R3D513 VARIABLE FNUSLBUS1*SNFSID13
R3D514 VARIABLE FNUSLBUS1*SNFSID14
R3D515 VARIABLE FNUSLBUS1*SNFSID15
D3D02 VARIABLE FNUSLBUS1*SNFSID11*SNFSID12
D3D03 VARIABLE FNUSLBUS1*SNFSID12
D3D04 VARIABLE FNUSLBUS1*SNFSID13
D3D05 VARIABLE FNUSLBUS1*SNFSID14
D3D05 VARIABLE FNUSLBUS1*SNFSID15
D3D01 VARIABLE FNUSLBUS1*SNFSID15
D3K11 VARIABLE FNUSLBUS1*SNFSRCK1
FILE: STCR VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 005

DKS1 BVARIABLE FNU$BUS1+SNF530X1
DKO1 BVARIABLE FNU$BUS1+SNF530X1
KDT11 BVARIABLE FNU$BUS1+SNF50T11
KDT12 BVARIABLE FNU$BUS1+SNF50T12
KDT13 BVARIABLE FNU$BUS1+SNF50T13
KDT14 BVARIABLE FNU$BUS1+SNF50T14
KDT15 BVARIABLE FNU$BUS1+SNF50T15
KDA11 BVARIABLE FNU$BUS1+SNF84011
KDA12 BVARIABLE FNU$BUS1+SNF84012
KDA13 BVARIABLE FNU$BUS1+SNF84013
KDA14 BVARIABLE FNU$BUS1+SNF84014
KDA15 BVARIABLE FNU$BUS1+SNF84015

-----------------------------

* BY FOR INTER LEVEL G0V

-----------------------------

KRR12 BVARIABLE FNU$G585+SNF531K2
KK512 BVARIABLE FNU$G585+SNF531K2
KKD12 BVARIABLE FNU$G585+SNF531K2
KKT11 BVARIABLE FNU$G585+SNF531K1
KKA21 BVARIABLE FNU$G585+SNF531K1

-----------------------------

* BY FOR L(2) G0S

-----------------------------

KRR2 BVARIABLE FNU$BUS2+SNF532X2
KRR2 BVARIABLE FNU$BUS2+SNF532X2
KRT2 BVARIABLE FNU$BUS2+SNF531R2
KRA2 BVARIABLE FNU$BUS2+SNF531R2
KRO2 BVARIABLE FNU$BUS2+SNF531R2
RDT21 BVARIABLE FNU$BUS2+SNF53121
RD521 BVARIABLE FNU$BUS2+SNF53121
RDT21 BVARIABLE FNU$BUS2+SNF53121
DKS2 BVARIABLE FNU$BUS2+SNF532X2
DKT2 BVARIABLE FNU$BUS2+SNF532X2
DKA2 BVARIABLE FNU$BUS2+SNF532X2
DKR2 BVARIABLE FNU$BUS2+SNF532X2
DKR2 BVARIABLE FNU$BUS2+SNF532X2
DKA2 BVARIABLE FNU$BUS2+SNF532X2

-----------------------------

* MACROS

-----------------------------


FILE: STCR  VS1JOB  A4

CONVERSATIONAL MONITOR SYSTEM

* BEGIN SIMULATION *

* SIMULATE *

* CPU #1 *

RMULT 3.5,7.9,11.13,15,17
CPU1 GENERATE ...X$WAXMP...F
START PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.1 CPU ID
ADVANCE X$=2.5
TRANSFER .X%=-EA2,WWW1,RRR1
RRE1 TRANSFER .X%=-1,NN11,NN11

* DATA IS IN DATA CACHE *

RIN11 ENTER RIC11 PUT TXN IN READ REQ BUFFER
USE MACRO DPP11,X$RDX1 SEARCH AND READ CACHE
LEAVE RID11 FREE BUFFER
ADVANCE ASCBUS
FINI MACRO TRANSFER ,STAR1 A NEW TXN

* DATA IS NOT IN CACHE *

RIN11 ENTER RIC11 PUT IN READ REQ BUFFER
USE MACRO DPP11,X$RDX1 SEARCH DIRECTORY
PRIORITY 0 RESET PRIORITY
SEND MACRO RID11,RIR1,LBUS1,X$BEX1,64$SORR1
TRANSFER ,COMR TO COMMON CODE FOR READ

* WRITE REQUEST TO CACHE *

WWW1 ENTER SII11 PUT TXN IN WRITE REQ BUFFER
USE MACRO DRPI1, XSREX WRITE DATA IN CACHE
  PRIORITY 0 RESET TXN PRIORITY
SEND MACRO SID11, SIR1, LB51, XSBEK1, BV$OR51

TRY1 ASSIGN 11, 0
  TRANSFER .X$PRV2, NTP11
USE MACRO RRPI1, X$SCS
  LEAVE SIR1
  BUFFER TEST E SIR1, 1
  ENTER SIR1
  TRANSFER TRY1
NTP11 ASSIGN 11, 0
  USE MACRO RRPI1, X$SCSJ
  TRANSFER .X$SHR, NTP11, SHR01
SHR01 ASSIGN 11, 0
SEND MACRO SIR1, SID11, LB51, XSBEK1, BV$OR511
USE MACRO DRPI1, XSREX2
SEND MACRO SID11, SOK1, LB51, XSBEK8, BV$OK51
SPLIT 1, COMM

FINI MACRO
  TRANSFER .STAR1
SHR1 ASSIGN 11, 0
  TEST E BV$1012, 1
  ENTER .011
  ENTER $10
  SEIZE .LB5
  ADVANCE XSBE1
  ASSIGN 4, .XSRE1
  RELEASE LB5
  LEAVE SIR1
  SPLIT 1, SHR21
USE MACRO DRPI1, XSREX2
SEND MACRO SID11, SOK1, LB51, XSBEK8, BV$OK51
SPLIT 1, COMM
FINI MACRO
  TRANSFER .STAR1
SHR21 ASSIGN 11, 0
USE MACRO DRPI2, XSREX1
  LEAVE SID12

-------------------------------
CPU = 2
-------------------------------
CPU2 GENERATE .X$MAXMP...F
STAR2 PRIORITY 9 SET HIGH P FOR NEW TXN
  MARK 2 ARRIVAL TIME
  ASSIGN 1.2 CPU 10
  ADVANCE X$CBUS
  TRANSFER .X$NREAD, WW2, RHR2
RRR2 TRANSFER .XSPIN1,MIN12,RRR12

* DATA IS IN DATA CACHE *

MIN12 ENTER RID12 PUT TN IN READ REQ BUFFER
USE MACRO DRP12,X$REX READ AND READ CACHE
LEAVE RID12 FREE BUFFER
ADVANCE X$CBUS
FINI MACRO TRANSFER .STAR2 A NEW TN

* DATA IS NOT IN CACHE *

MIN12 ENTER RID12 PUT TN IN READ REQ BUFFER
USE MACRO DRP12,X$REX SEARCH DIRECTORY
PRIORITY 0 RESET PRIORITY
SEND MACRO RID12,RIR1,LIB51,X$AX1,6,B$DR11
TRANSFER .COMR TO COMMON CODE FOR READ

* WRITE REQUEST TO CACHE *

WWW2 ENTER SID12 PUT TN IN WRITE REQ BUFFER
USE MACRO DRP12,X$REX WRITE DATA IN CACHE
PRIORITY 0 RESET TN PRIORITY
SEND MACRO SID12,SIR1,LIB51,X$AX1,6,B$SR11

TRY2 ASSIGN 11,0 TRANSFER .X$PRV2,,NTP12 USE MACRO RRP1,X$CS
LEAVE SIR1 BUFFER
TEST E SR1,1 ENTER SIR1 TRANSFER .TRY2
NTP12 ASSIGN 11,0 USE MACRO RRP1,X$SCS TRANSFER .X$SHR,SHR12,SHR02
SHR02 ASSIGN 11,0 SEND MACRO SIR1,SID12,LIB51,X$AX1,6,B$SR512
FILE: STCR VP1 JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 010

USE MACRO DRP12, X$RDEX2
SEND MACRO SID12, S0K1, LBUS1, X$BEXA, BV$DKS1
SPLIT 1, COMW
FINI MACRO TRANSFER, STAR2

SHR12 ASSIGN 11, 0
TEST E BV$D203.1
ENTER SID12
ENTER SID13
SEIZE LBUS1
ADVANCE X$BEX1
ASSIGN 4, X$BEX1
RELEASE LBUS1
LEAVE SIR1
SPLIT 1, SHR22
USE MACRO DRP12, X$RDEX2
SEND MACRO SID12, S0K1, LBUS1, X$BEXA, BV$DKS1
SPLIT 1, COMW
FINI MACRO TRANSFER, STAR2

SHR22 ASSIGN 11, 0
USE MACRO DRP13, X$REX1
LEAVE SID13

TERMINATE

***************

* CPJ #3 *

***************

CPU3 GENERATE ..., X$MAXMP, ...

 Starr PRIORITY 9 SET HIGH P FOR NEW TXN
 MARK 2 ARRIVAL TIME
 ASSIGN 1,3 CPU ID
 ADVANCE X$CBUS
 TRANSFER ..., X$READ, WWW3, RRR3
 RRR3 TRANSFER ..., X$PIN1, MIN13, RRR13

---------

* DATA IS IN DATA CACHE *

---------

FIN-3 ENTER RIO13 SET TXN IN READ REQ BUFFER
USE MACRO DRP13, X$RDEX1 SEARCH AND READ CACHE
LEAVE RIO13 FREE BUFFER
ADVANCE X$CBUS
FINI MACRO TRANSFER, STAR3 A END TXN

---------

* DATA IS NOT IN CACHE *

---------
FILE: STOR VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

92

****NI013 ENTER RID13 PUT IN READ REQ BUFFER****
USE MACRO DRP13,XSREX SEARCH DIRECTORY
PRIORITY 0 RESET PRIORITY
SEND MACRO RID13,RIR1,LSUX1,XS8EX1,BSVDSR1
TRANSFER .CCMR TO COMMON CODE FOR READ

# WRITE REQUEST TO CACHE#
#

****WWW3 ENTER SID13 PUT TXN IN WRITE REQ BUFFER****
USE MACRO DRP13,XSREX WRITE DATA IN CACHE
PRIORITY 0 RESET TXN PRIORITY
SEND MACRO SID13,SIR1,LSUX1,XS8EX1,BSVDSR1

TRY3 ASSIGN 11.0
TRANSFER .XS5=E/2,NTP13
USE MACRO RRP1,XS5CS LEAVE SIR1
BUFFER TEST E SIR1.1
ENTER SIR1
TRANSFER .TRY3

NTP13 ASSIGN 11.0
USE MACRO RRP1,XS5CSU TRANSFER .XS5=E,SHR13,SHR03

SHR03 ASSIGN 11.0
SEND MACRO SIR1,SID13,LSUX1,XS8EX1,BSVDSR13
USE MACRO DRP13,XS8DEX2
SEND MACRO SID13,SOX1,LSUX1,XS8EX8,BSVDSK1
SPLIT 1.COM#

FINI MACRO
TRANSFER .STAR3

SHR13 ASSIGN 11.0
TEST E BS5D3D4,1
ENTER S1-13
ENTER SID14
SEIZE LSUX1
ADVANCE XS8EX1
ASSIGN d=XS8EX1
RELEASE LSUX1
LEAVE SIR1
SPLIT 1.5-223

USE MACRO DRP13,XS8DEX2
SEND MACRO SID13,SOX1,LSUX1,XS8EX8,BSVDSK1
SPLIT 1.COM#
FILE: STCR VS: JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 012

FINI MACRO
TRANSFER .STAR3

SHR23 ASSIGN 11.0
USE MACRO DRP14.X$REX1
LEAVE SID14
TERMINATE

******************************
* CPU #4
* ******************************

******************************
CPU4 GENERATE ...X$MAXMP...F
STAR4 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.4 CPU ID
ADVANCE X$CBUS
TRANSFER .X$READ.WWW4,RRR4
RRR4 TRANSFER .X$PIN1,NIN14,RIN14

******************************
* DATA IS IN DATA CACHE*
**************************************************************
RIN14 ENTER RID14 PUT TXN IN READ REQ BUFFER
USE MACRO DRP14.X$REX1 SEARCH AND READ CACHE
LEAVE RID14 FREE BUFFER
ADVANCE X$CBUS
FINI MACRO TRANSFER .STAR4 A NEW TXN

******************************
* DATA IS NOT IN CACHE*
**************************************************************
NIN14 ENTER RID14 PUT IN READ REQ BUFFER
USE MACRO DRP14.X$REX SEARCH DIRECTORY
SENO MACRO RID14,RIR1,LBUS1.X$REX1,BS$RR1
TRANSFER .COMR TO COMMON CODE FOR READ

******************************
* WRITE REQUEST TO CACHE*
**************************************************************
WWW4 ENTER SID14 PUT TXN IN WRITE REQ BUFFER
USE MACRO DRP14.X$REX WRITE DATA IN CACHE
FILE: SCR VS1J00 A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 013

PRIORITY 0

RESET TXN PRIORITY

SEND MACRO SID14,SIR1,LIBS1,X$BEX1,BY$DRS1

TRY4 ASSIGN 11.0
TRANSFER .X$PRV2.,NTP14

USE MACRO $D31,X$SCS
LEAVE SIR1
BUFFER SR1,1
ENTER SIR1
TRANSFER TRY4

NTPI4 ASSIGN 11.0
USE MACRO ARPI1,X$SCSU
TRANSFER .X$SR4,SHR14,SHR04

SHR04 ASSIGN 11.0
SEND MACRO SID14,LIBS1,X$BEX1,BY$DRS14

USE MACRO DRPI4.X$RDEX2
SEND MACRO SID14,SOK1,LIBS1,X$BEXB,BY$CKS1
SPLIT 1.COMW
FINI MACRO
TRANSFER .STAR4

SHR14 ASSIGN 11.0
"TEST" B:50405.1
ENTER SID14
ENTER SID15
SEIZE LIBS1
ADVANCE X$BEX1
ASSIGN 4.,X$BEX1
RELEASE LIBS1
LEAVE SIR1
SPLIT 1.SHR24

USE MACRO DRPI4.X$RDEX2
SEND MACRO SID14,SOK1,LIBS1,X$BEXB,BY$DKS1
SPLIT 1.COMW
FINI MACRO
TRANSFER .STAR4

SHR24 ASSIGN 11.0
USE MACRO DRPI5,X$BEX1
LEAVE SID15
TERMINATE

***************

* CPJ #5
*
***************

CPUS GENERATE ...X$MAXMP,...F

STARS PRIORITY 9

SET HIGH P FOR NEW TXN

MASK 2

ARRIVAL TIME

ASSIGN 1,5

CPU ID

ADVANCE X$CBUS

TRANSFER .X$NRERAD,WWW5.RRR5
FILE: STCR VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 014

RRRS TRANSFER .X$PIN1,NIN15,RIN15

-------------------------------
* DATA IS IN DATA CACHE *
-------------------------------

RIN15 ENTER RID15 PUT TXN IN READ REQ BUFFER

USE MACRO DRS15,X$DEX1 SEARCH AND READ CACHE

LEAVE RID15 FREE BUFFER

ADVANCE X$CBSUS

FINI MACRO TRANSFER .STARS A NEW TXN

-------------------------------
* DATA IS NOT IN CACHE *
-------------------------------

NIN15 ENTER RID15 PUT TXN IN READ REQ BUFFER

USE MACRO DRS15,X$REX SEARCH DIRECTORY

PRIORITY 0 RESET PRIORITY

SEND MACRO RID15,RIR1,LBUS1,X$BEX1,B5$DR1

TRANSFER .COMR TO COMMON CODE FOR READ

-------------------------------
* WRITE REQUEST TO CACHE *
-------------------------------

WWW ENTER SID15 PUT TXN IN WRITE REQ BUFFER

USE MACRO DRS15,X$RXX WRITE DATA IN CACHE

PRIORITY 0 RESET TXN PRIORITY

SEND MACRO SID15,SIR1,LBUS1,X$BEX1,B5$DRS1

TRYS ASSIGN 11.0

TRANSFER .X$PRV2,,NTP15

USE MACRO RRP1,X$SCS

LEAVE SIR1

SUFFER

TEST E SRI1.1

ENTER SIR1

TRANSFER .TRYS

NTP15 ASSIGN 11.0

USE MACRO RRP1,X$SCSU

TRANSFER .X$SR15,SHROS

SHROS ASSIGN 11.0
FILE: STCR  VS1-J03  A4  CONVERSATIONAL MONITOR SYSTEM  PAGE 015

SEND MACRO S1R1,SID15,LE=51,X$BEK1,BV$RODS15
USE MACRO DRP15,X$ROEX2
SEND MACRO SID15,SOX1,LE=51,X$BEK8,BV$D0S1
SPLIT 1.COM
FINI MACRO
TRANFER .STTX25
SRR15 ASSIGN 11.0
TEST E BV$C5C1.1
ENTER SID15
ENTER SID11
SEIZE LB=51
ADVANCE X$BEK1
ASSIGN 44.73EEA1
RELEASE LEU51
LEAVE S1R1
SPLIT 1.S025
USE MACRO DRP15,X$ROEX2
SEND MACRO SID15,SOX1,LE=51,X$BEK8,BV$D0S1
SPLIT 1.COM
FINI MACRO
TRANFER .STRX5
SRR20 ASSIGN 11.0
USE MACRO DRP11,X$REK1
LEAVE SID11
TERMINATE

* COMMON CODE FOR READ REQUEST *

COMM ASSIGN 11.0
USE MACRO RRP1,X$SCS
TRANSFER .X$PRV1,NTP.
LEAVE R1R1
BUFFER TEST E BV$RR1.1
ENTER R1R1
TRANSFR .COMM
MTPV ASSIGN 11.0
SEND MACRO R1R1,ROK1,LE=51,X$BEK1,BV$RKR1
USE MACRO KRP1,X$KEK
SEND MACRO ROK1,RIR2,GB=5,X$BEK1,BV$KRR12
USE MACRO KRP2,X$KEK
SEND MACRO RIK2,RIR2,GB=5,X$BEK1,BV$KRR2
USE MACRO RRP2,X$REK

*
USE MACRO DRP11,XSRPLR

SPLIT 1.OVL11
TEST E BV$DRK01.1
ENTER OIR1
ENTER OOK1
SEIZE LBUS1
ADVANCE X$8EX1
ASSIGN 4+,X$8EX1
RELEASE LBUS1
LEAVE TID11
SPLIT 1.OVL1
USE MACRO RRP1,X$SCU
LEAVE OIR1

TERMINATE

OVL11 ASSIGN 11.0
ADVANCE X$CBUS
FINI MACRO
TRANSFER ,STAR1

**************

* RT STORE INTO D12 *

**************

WWW12 ASSIGN 11.0
SEND MACRO TIK1,TID12,L3,S1,X$8EX8,BV$KDT12
USE MACRO DRP12,XSRPLR

SPLIT 1.OVL12
TEST E BV$DRK01.1
ENTER OIR1
ENTER OOK1
SEIZE LBUS1
ADVANCE X$8EX1
ASSIGN 4+,X$8EX1
RELEASE LBUS1
LEAVE TID12
SPLIT 1.OVL1
USE MACRO RRP1,X$SCU
LEAVE OIR1

TERMINATE

OVL12 ASSIGN 11.0
ADVANCE X$CBUS
FINI MACRO
TRANSFER ,STAR2

**************

* RT STORE INTO D13 *

**************
CONVERSATIONAL MONITOR SYSTEM

FILE: SICR  VS1JOB  A4

---

COMM ASSIGN 11.0
USE MACRO KRP1,X$KEK
SEND MACRO SDK1,S1K2,GBU5,X$BEX8,BV$KKS12
USE MACRO KRP2,X$KEK
SEND MACRO S1K2,S1R2,LBUS2,X$BEX8,BV$KRS2
USE MACRO RRPI,X$KEK
SEND MACRO S1R2,S1D21,LBUS2,X$BEX8,BV$RD521
USE MACRO QRPI,X$DEX2
SEND MACRO S1D21,ADK2,LBUS2,X$BEX1,BV$DKS2

---

* AC* FROM L(2) TO L(1)

---

ACK21 ASSIGN 11.0
USE MACRO KRP2,X$KEK
SEND MACRO ADK2,AK1,GBU5,X$BEX1,BV$KKA21
USE MACRO KRP1,X$KEK
SPLIT 1,FNSWICH
TERMINATE

---

* ACK HANDLED BY D11

---

AAA11 ASSIGN 11.0
SEND MACRO AK1,AID11,LBUS1,X$BEX1,BV$ADA11
USE MACRO PRPI1,X$DECR
TRANSFER .X$SCNT.,CNFI1
LEAVE AID11
TERMINATE
CNFI1 ASSIGN 11.0
SEND MACRO AID11,AIR1,LBUS1,X$BEX1,BV$DRA1

---
FILE: SCR VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

USE MACRO RRP1, X$SCU
LEAVE AIR1

TERMINATE

* ACK-HANDLED BY D12 *

*******

AAA12 ASSIGN 11, 0
SEND MACRO AIK1, AID12, LBUS1, X$SEX1, BY$KDA12
USE MACRO DRP12, X$DECR
TRANSFER .X$CNT,,CNT12
LEAVE AID12
TERMINATE

CNT12 ASSIGN 11, 0
SEND MACRO AID12, AIR1, LBUS1, X$SEX1, BY$DRA1
USE MACRO RRP1, X$SCU
LEAVE AIR1

TERMINATE

* ACK-HANDLED BY D13 *

*******

AAA13 ASSIGN 11, 0
SEND MACRO AIK1, AID13, LBUS1, X$SEX1, BY$KDA13
USE MACRO DRP13, X$DECR
TRANSFER .X$CNT,,CNT13
LEAVE AID13
TERMINATE

CNT13 ASSIGN 11, 0
SEND MACRO AID13, AIR1, LBUS1, X$SEX1, BY$DRA1
USE MACRO RRP1, X$SCU
LEAVE AIR1

TERMINATE

* ACK-HANDLED BY D14 *

*******

AAA14 ASSIGN 11, 0
SEND MACRO AIK1, AID14, LBUS1, X$SEX1, BY$KDA14
USE MACRO DRP14, X$DECR
TRANSFER .X$CNT,,CNT14
LEAVE A1D14
TERMINATE
CNT14 ASSIGN 11,0
SEND MACRO A1D14, AIR1, LBUS1, X$BEX1, BV$DRA1
USE MACRO RRP1, X$SCU
LEAVE AIR1
TERMINATE

************
* ACK HANDLED BY D15 *
*
************

AAA15 ASSIGN 11,0
SEND MACRO A.41, A1D15, LBUS1, X$BEX1, BV$KDA15
USE MACRO DRP15, X$DECR
TRANSFER .X$CNT,,CNT15
LEAVE A1D15
TERMINATE
CNT15 ASSIGN 11,0
SEND MACRO A1D15, AIR1, LBUS1, X$BEX1, BV$DRA1
USE MACRO RRP1, X$SCU
LEAVE AIR1
TERMINATE

************
* SIMULATION CONTRL *
*
************

GENERATE X$TIMER
TERMINATE 1
START 1
END

$5 END
APPENDIX (III): The "CC/PTB/C" Program
// TAR1 JOB TAR,
// PROFILE="DEFER",
// TIME=5
// *PASSWORD SCUBA
// GPSS PROC
// C EXEC PGM=DAG01,TIME=ATLIMIT
// STEPLIB DD DSN=PDUC1.LIBRARY,GPSS.LOAD,DISP=SHR
// OUTPUT DD SYSOUT=PROFILE=PRINT,DCB=BLKSIZE=931
// DINTER DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=1382
// DSMTAB DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=7:12
// DREPTGEN DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=820
// DINTWORK DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=2680
// PEND
// STEP1 EXEC GPSS,PARM=C,TLIMIT=9
// INPUT DD *
// REALLOCATE F:\W,6,QUE,10,FAC,50,BVR,200,BLO,2000,VAR,50
// REALLOCATE FSV,50,HSV,10,COM,39968

**************************************************
*** CC/PTB/C
**************************************************

- **TRANSACTION PARAMETER USAGE**
  - **P1** CPU ID
  - **P2** TXN ARRIVAL TIME
  - **P3** TXN COMPL TIME
  - **P4** TXN EXEC TIME
  - **P5** DEVICE ID
  - **P11** DUMMY

**************************************************
*** MODEL COMPONENTS

- **BUSES**: GBUS, LBUS...
- **CACHES**: D11,...,D15
- **LEVEL CONTROLS**: A1,A2
- **REQ PROCESSES**: #2
- **DEVICES**: D21
- **STORAGE**: RI, RO
- **STORAGE**: SI, SO
- **STORAGE**: TI, TO
- **STORAGE**: AI, AO
- **STORAGE**: CI, CO

**************************************************
*** MODEL PARAMETERS
FILE: CCC VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

1. STAR1/2, STAR2/3, STAR3/4, STAR4/5, STAR5

   WICH FUNCTION P5.05
   1. WICH1/2, WICH2/3, WICH3/4, WICH4/5, WICH5

2. UNIT FUNCTION P5.05
   1. WDD1/2, WDD2/3, WDD3/4, WDD4/5, WDD5

2. UNIT FUNCTION P5.05
   1. RDD1/2, RDD2/3, RDD3/4, RDD4/5, RDD5

LOAD FUNCTION RN1.05
   1. 1/4, 2/6, 3/8, 4/1, 5

1. WICH FUNCTION P5.05
   1. AAA1/2, AAA2/3, AAA3/4, AAA4/5, AAA5

* * * * * * * * * * *

* STORAGE FOR L(1)
* CACHES
* *
* * * * * * * * * * *

STORAGE SS6ID11,10/SS6ID11,10/SS6ID11,10/SS6ID11,10
STORAGE SS6ID12,10/SS6ID12,10/SS6ID12,10/SS6ID12,10
STORAGE SS6ID13,10/SS6ID13,10/SS6ID13,10/SS6ID13,10
STORAGE SS6ID14,10/SS6ID14,10/SS6ID14,10/SS6ID14,10
STORAGE SS6ID15,10/SS6ID15,10/SS6ID15,10/SS6ID15,10

* * * * * * * * * * *

* STORAGE FOR DEVICES
* *
* * * * * * * * * * *

STORAGE SS6R21,10/SS6R21,10/SS6R21,10

* * * * * * * * * * *

* STORAGE FOR REQ PROC
* *
* * * * * * * * * * *

STORAGE SS6R11,10/SS6R11,10/SS6R11,10/SS6R11,10
STORAGE SS6R12,10/SS6R12,10/SS6R12,10/SS6R12,10

* * * * * * * * * * *

* STORAGE FOR KI
* *
* * * * * * * * * * *

STORAGE SS6OK1,10/SS6OK1,10/SS6OK1,10/SS6OK1,10
FILE: CCC VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 004

* STORAGE FOR K2,K3,K4 *

-----------------------------

STORAGE $S$R1K2,10/$S$S1K2,0/$S$TIK2,10/$S$AIK2,10/$S$QIK2,10
STORAGE $S$R0K2,10/$S$S0K2,10/$S$TOK2,10/$S$ADOK2,10/$S$DOK2,10

-----------------------------

* BOOLEAN VARIABLES *

-----------------------------

* BY FOR READ-THROUGH *

-----------------------------

ATOK2 BVARIABLE FNUSGBUS+SNFSTIK1

-----------------------------

* BY FOR L(1) *

-----------------------------

CR01 BVARIABLE FNUSLBUS1+SNFSR01
CR01 BVARIABLE FNUSLBUS1+SNFSR01
CR02 BVARIABLE FNUSLBUS1+SNFSR02
CR02 BVARIABLE FNUSLBUS1+SNFSR02
CR03 BVARIABLE FNUSLBUS1+SNFSR03
CR03 BVARIABLE FNUSLBUS1+SNFSR03
CR04 BVARIABLE FNUSLBUS1+SNFSR04
CR04 BVARIABLE FNUSLBUS1+SNFSR04
CR05 BVARIABLE FNUSLBUS1+SNFSR05
CR05 BVARIABLE FNUSLBUS1+SNFSR05
CR06 BVARIABLE FNUSLBUS1+SNFSR06
CR06 BVARIABLE FNUSLBUS1+SNFSR06
CR07 BVARIABLE FNUSLBUS1+SNFSR07
CR07 BVARIABLE FNUSLBUS1+SNFSR07
CR08 BVARIABLE FNUSLBUS1+SNFSR08
CR08 BVARIABLE FNUSLBUS1+SNFSR08
CR09 BVARIABLE FNUSLBUS1+SNFSR09
CR09 BVARIABLE FNUSLBUS1+SNFSR09
CR10 BVARIABLE FNUSLBUS1+SNFSR10
CR10 BVARIABLE FNUSLBUS1+SNFSR10
CR11 BVARIABLE FNUSLBUS1+SNFSR11
CR11 BVARIABLE FNUSLBUS1+SNFSR11
CR12 BVARIABLE FNUSLBUS1+SNFSR12
CR12 BVARIABLE FNUSLBUS1+SNFSR12
CR13 BVARIABLE FNUSLBUS1+SNFSR13
CR13 BVARIABLE FNUSLBUS1+SNFSR13
CR14 BVARIABLE FNUSLBUS1+SNFSR14
CR14 BVARIABLE FNUSLBUS1+SNFSR14
CR15 BVARIABLE FNUSLBUS1+SNFSR15
CR15 BVARIABLE FNUSLBUS1+SNFSR15
CR16 BVARIABLE FNUSLBUS1+SNFSR16
CR16 BVARIABLE FNUSLBUS1+SNFSR16
CR17 BVARIABLE FNUSLBUS1+SNFSR17
CR17 BVARIABLE FNUSLBUS1+SNFSR17
CR18 BVARIABLE FNUSLBUS1+SNFSR18
CR18 BVARIABLE FNUSLBUS1+SNFSR18
CR19 BVARIABLE FNUSLBUS1+SNFSR19
CR19 BVARIABLE FNUSLBUS1+SNFSR19
CR20 BVARIABLE FNUSLBUS1+SNFSR20
CR20 BVARIABLE FNUSLBUS1+SNFSR20

-----------------------------

DK01 BVARIABLE FNUSLBUS1+SNFSRDK1
DK01 BVARIABLE FNUSLBUS1+SNFSRDK1
DK02 BVARIABLE FNUSLBUS1+SNFSRDK2
DK02 BVARIABLE FNUSLBUS1+SNFSRDK2
DK03 BVARIABLE FNUSLBUS1+SNFSRDK3
DK03 BVARIABLE FNUSLBUS1+SNFSRDK3
DK04 BVARIABLE FNUSLBUS1+SNFSRDK4
DK04 BVARIABLE FNUSLBUS1+SNFSRDK4
DK05 BVARIABLE FNUSLBUS1+SNFSRDK5
DK05 BVARIABLE FNUSLBUS1+SNFSRDK5
DK06 BVARIABLE FNUSLBUS1+SNFSRDK6
DK06 BVARIABLE FNUSLBUS1+SNFSRDK6
DK07 BVARIABLE FNUSLBUS1+SNFSRDK7
DK07 BVARIABLE FNUSLBUS1+SNFSRDK7
DK08 BVARIABLE FNUSLBUS1+SNFSRDK8
DK08 BVARIABLE FNUSLBUS1+SNFSRDK8
DK09 BVARIABLE FNUSLBUS1+SNFSRDK9
DK09 BVARIABLE FNUSLBUS1+SNFSRDK9
DK10 BVARIABLE FNUSLBUS1+SNFSRDK10
DK10 BVARIABLE FNUSLBUS1+SNFSRDK10
DK11 BVARIABLE FNUSLBUS1+SNFSRDK11
DK11 BVARIABLE FNUSLBUS1+SNFSRDK11
DK12 BVARIABLE FNUSLBUS1+SNFSRDK12
DK12 BVARIABLE FNUSLBUS1+SNFSRDK12
DK13 BVARIABLE FNUSLBUS1+SNFSRDK13
DK13 BVARIABLE FNUSLBUS1+SNFSRDK13
DK14 BVARIABLE FNUSLBUS1+SNFSRDK14
DK14 BVARIABLE FNUSLBUS1+SNFSRDK14
DK15 BVARIABLE FNUSLBUS1+SNFSRDK15
DK15 BVARIABLE FNUSLBUS1+SNFSRDK15
DK16 BVARIABLE FNUSLBUS1+SNFSRDK16
DK16 BVARIABLE FNUSLBUS1+SNFSRDK16
DK17 BVARIABLE FNUSLBUS1+SNFSRDK17
DK17 BVARIABLE FNUSLBUS1+SNFSRDK17
DK18 BVARIABLE FNUSLBUS1+SNFSRDK18
DK18 BVARIABLE FNUSLBUS1+SNFSRDK18
DK19 BVARIABLE FNUSLBUS1+SNFSRDK19
DK19 BVARIABLE FNUSLBUS1+SNFSRDK19
DK20 BVARIABLE FNUSLBUS1+SNFSRDK20
DK20 BVARIABLE FNUSLBUS1+SNFSRDK20

-----------------------------
* BY FOR L(2) OPS

* MACROS

* MACRO -USE
* #A FACILITY
* #B USAGE TIME

USE START_MACRO
SEIZE #A
ADVANCE #B
ASSIGN 4+,#B
RELEASE #A
END_MACRO

* BY FOR INTER LEVEL CON*
FILE: CCC VSJOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 006

MACRO - SEND

#A FROM
#B TO
#C VIA
#D TRANSIT TIME
#E BY FOR SEND OP

***************

SEND STARTMACRO
TEST E #E,1
ENTER #B
SEIZE #C
ADVANCE #D
ASSIGN #D,#D
RELEASE #C
LEAVE #A
ENDMACRO

***************

MACRO - SND

***************

SND STARTMACRO
TEST E #D,1
ENTER #A
SEIZE #B
ADVANCE #C
ASSIGN #D,#C
RELEASE #B
ENDMACRO

***************

MACRO - FINI

***************

FINI STARTMACRO
MARK 3
SAVEVALUE NTXN+1
SAVEVALUE SUM+VSTXNX
SAVEVALUE SUM+VSTXNN
SAVEVALUE SUM+VSTXNT
SAVEVALUE MRESP,VSMRESP
ASSIGN 2.0
ASSIGN 3.0
ASSIGN 4.0
ENDMACRO

***************

BEGIN SIMULATION
FILE: CCC VS1JDB A4 CONVERSATIONAL MONITOR SYSTEM

SIMULATE

CPU #1

RMULT 3, 5, 7, 9, 11, 13, 15, 17

CPU1 GENERATE 
STAR1 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.1 CPU ID
TRANSFER X$READ,WWW1,READ

CPU #2

CPU2 GENERATE 
STAR2 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.2 CPU ID
TRANSFER X$READ,WWW1,READ

CPU #3

CPU3 GENERATE 
STAR3 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.3 CPU ID
TRANSFER X$READ,WWW1,READ

CPU #4

CPU4 GENERATE 
STAR4 PRIORITY 9 SET HIGH P FOR NEW TXN
MARK 2 ARRIVAL TIME
ASSIGN 1.4 CPU ID
TRANSFER X$READ,WWW1,READ
FILE: CCC  VS:JOB  A4  CONVERSATIONAL MINIOR SYSTEM  PAGE 008

*************************************************

CPU #5

*************************************************

CPUs GENERATE .X$4.X$8P,...F
STARS PRIORITY 9  SET HIGH P FOR NEW TXT.
MARK 2  ARRIVAL TIME
ASSIGN 1.5  CPU ID
TRANSFER .X$READ,WWW1,READ

*************************************************

DATA TO BE READ

*************************************************

READ ASSIGN 11,0
ASSIGN 5,FSLOAD
TRANSFER .FSRUNIT

*************************************************

DEVICE D11

*************************************************

RDO11 ASSIGN 5,1
SEND MACRO RID11,LIBUS1,X$BEX1,BV$RD11
TRANSFER .X$PIN1,NIN11,RIN11
RIN11 ASSIGN 11,0
USE MACRO DRP11,X$RDEX1
SEIZE LIBUS1
ADVANCE X$BEX1
ASSIGN 4+,X$BEX1
RELEASE LIBUS1
LEAVE RID11
Fini MACRO
SPLIT 1,FSWCHST
TERMIALE
NIN11 ASSIGN 11,0
PRIORITY 0
USE MACRO DRP11,X$BEX
SEND MACRO RID11,ROK1,LIBUS1,X$BEX1,UV$DWR1
TRANSFER .COMM

*************************************************

DEVICE D12

*
FILE: CCC   VS1JOB   AA   CONVERSATIONAL MONITOR SYSTEM   PAGE 010

SMO MACRO   RID14, LBUS1, X$BEX1, BV$RD14
TRANSFER   .XSPIN1, MIN14, RIN14
RIN14 ASSIGN   11.0
USE MACRO   DP14, X$RD1X1
SEIZE   LBUS1
ADVANCE   X$BEX1
ASSIGN   4-, X$BEX1
RELEASE   LBUS1
LEAVE   RID14
FINI MACRO   SPLIT   1, FNSWCHST
TERMINATE
RIN14 ASSIGN   11.0
PRIORITY   0
USE MACRO   DP14, X$REX
SEND MACRO   RID14, ROK1, LBUS1, X$BEX1, BV$DKR1
TRANSFER   , COMR

**************************************************

*  *
*  DEVICE DIS  *
*  *
**************************************************

RIN15 ASSIGN   5.5
SMO MACRO   RID15, LBUS1, X$BEX1, BV$RD15
TRANSFER   .XSPIN1, MIN15, RIN15
RIN15 ASSIGN   11.0
USE MACRO   DP15, X$RD1X1
SEIZE   LBUS1
ADVANCE   X$BEX1
ASSIGN   4-, X$BEX1
RELEASE   LBUS1
LEAVE   RID15
FINI MACRO   SPLIT   1, FNSWCHST
TERMINATE
RIN15 ASSIGN   11.0
PRIORITY   0
USE MACRO   DP15, X$REX
SEND MACRO   RID15, ROK1, LBUS1, X$BEX1, BV$DKR1
TRANSFER   , COMR

**************************************************

*  *
*  DATA TO BE WRITTEN IS IN LEVEL 1  *
*  *
**************************************************

WNN1 ASSIGN   11.0
FILE: CCC VS 1 JOB A4

CONVERSATIONAL MONITOR SYSTEM

ASSIGN 5.FN$LOAD
TRANSFER .FN$UNIT

**********************

* DEVICE D11

**********************

WDD11 ASSIGN 5.1
SMD MACRO SID11, LBUS1, X$BEX1, BV$RD511
USE MACRO DRP11, X$RDEX1
PRIORIT Y 0
SEND MACRO SID11, SOK1, LBUS1, X$BEX8, BV$SDK51
SPLIT 1, COMW
FINI MACRO SPLIT 1, FN$WCHST
TERMINATE

**********************

* DEVICE D12

**********************

WDD12 ASSIGN 5.2
SMD MACRO SID12, LBUS1, X$BEX1, BV$RD512
USE MACRO DRP12, X$RDEX1
PRIORIT Y 0
SEND MACRO SID12, SOK1, LBUS1, X$BEX8, BV$SDK51
SPLIT 1, COMW
FINI MACRO SPLIT 1, FN$WCHST
TERMINATE

**********************

* DEVICE D13

**********************

WDD13 ASSIGN 5.3
SMD MACRO SID13, LBUS1, X$BEX1, BV$RD513
USE MACRO DRP13, X$RDEX1
PRIORIT Y 0
SEND MACRO SID13, SOK1, LBUS1, X$BEX8, BV$SDK51
SPLIT 1, COMW
FINI MACRO SPLIT 1, FN$WCHST
TERMINATE
FILE: CCC VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 015

OVF13 TRANSFER  .OVL1

********************

* RT STORE INTO D14 *

********************

WWW14 ASSIGN 11.0
SEND MACRO TIK1,TID14,LIBUS1,X$BEX8,BV$KDT14
USE MACRO DRI14,X$RPLR
SEND MACRO TID14,ODK1,LIBUS1,X$BEX1,BV$DKO1
SPLIT 1.OVF14
FINI MACRO SPLIT 1,FN$WCHST
TERMINATE

OVF14 TRANSFER  .OVL1

********************

* RT STORE INTO D15 *

********************

WWW15 ASSIGN 11.0
SEND MACRO TIK1,TID15,LIBUS1,X$BEX8,BV$KDT15
USE MACRO DRI15,X$RPLR
SEND MACRO TID15,ODK1,LIBUS1,X$BEX1,BV$DKO1
SPLIT 1.OVF15
FINI MACRO SPLIT 1,FN$WCHST
TERMINATE

OVF15 TRANSFER  .OVL1

********************

* HANDLE OVF FROM L(1) *

********************

OVL1 ASSIGN 11.0
USE MACRO KRP1,X$KEK
FILE: CCC  VS1JOB  A4

CONVERSATIONAL MONITOR SYSTEM

SEND MACRO  DD41, DIA2, GBUS, X$BEX1, BV$KQ012
USE MACRO   KRP2, X$KEX
SEND MACRO  DIA2, DIA2, LBUS2, X$BEX1, BV$KRO2
USE MACRO   RRP2, X$REX
LEAVE        DIA2
TERMINATE

* * * COMMON CODE FOR STOP-BEFORE
  * * *

COM  ASSIGN  11.C
USE MACRO  KRP1, X$KEX
SEND MACRO  SCK1, SCK2, GBUS, X$BEX8, BV$KQ12
USE MACRO   KRP2, X$KEX
SEND MACRO  SCK1, SCK2, LBUS2, X$BEX8, BV$KRS2
USE MACRO   RRP2, X$REX
SEND MACRO  SCK2, SCK2, LBUS2, X$BEX8, BV$RDS21
USE MACRO   DRP21, X$DEX2
SEND MACRO  SCK2, SCK2, LBUS2, X$BEX1, BV$DKS2

* * * ACK FROM L(2) TO L(1)
  * * *

USE MACRO   KRP2, X$KEX
SEND MACRO  ACK2, AK1, GBUS, X$BEX1, BV$KQA21
USE MACRO   KRP1, X$KEK

SPLIT        1, FNSWICHA
TERMINATE
***************
* ACK HANDLED BY D11 *
***************

AAA1 ASSIGN 11,0
SEND MACRO AIK1, AID11, LBUS1, XSBEX1, BSXDA11
USE MACRO DRP11, XSERX
   LEAVE AID11
   TERMINATE

***************
* ACK HANDLED BY D12 *
***************

AAA12 ASSIGN 11,0
SEND MACRO AIK1, AID12, LBUS1, XSBEX1, BSXDA12
USE MACRO DRP12, XSERX
   LEAVE AID12
   TERMINATE

***************
* ACK HANDLED BY D13 *
***************

AAA13 ASSIGN 11,0
SEND MACRO AIK1, AID13, LBUS1, XSBEX1, BSXDA13
USE MACRO DRP13, XSERX
   LEAVE AID13
   TERMINATE

***************
* ACK HANDLED BY D14 *
***************

AAA14 ASSIGN 11,0
SEND MACRO AIK1, AID14, LBUS1, XSBEX1, BSXDA14
USE MACRO DRP14,X$REX
     LEAVE AID14
     TERMINATE

                                     * ACK HANDLED BY D15 *
                                     *
                                     *

AAA15 ASSIGN 11.0
SEND MACRO A1K1,AID15,LBUS1,X$BEX1,BV$KDA15
USE MACRO DRP15,X$REX
     LEAVE AID15
     TERMINATE

                                     * SIMULATION CONTROL *
                                     *
                                     *

     GENERATE X$TIMER
     TERMINATE 1
     START 1
     END

SS END
APPENDIX (IV): The "CC/PTB/P" Program
FILE: CCP .S1.JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 001

//TAR1 JOB TAR,
//PROFILE="DEFER",
//TIME=5
//PASS WORD SCJEBA
//GPSS PROC
//EXEC COM=32701, TIME=ATLIMIT
//STEP 1 GSS=FLACK.LIBRARY.GPSS.LOAD.DISP=SHR
//OUTPUT GSS=PRINT,DCB=BLKSIZE=931
//DINTERD DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=1800
//DSYMTAB DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=7112
//DREVGEN DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=800
//DINTERPROC DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=2600
//MEMO
//STEP1 EXEC GPSS.PARM=C,TLIMIT=9
//DINPU T DD *
REALLOCATE FUG.QUE,10,FAC,50,BVR,200,BLO,2000,VAR,50
REALLOCATE FSV,50,HSV,10,COM,39988

**************************************************************
* TAN PARM USAGE
**************************************************************
* P1 CPU ID
* P2 TAN ARRIVAL TIME
* P3 TAN COMPL TIME
* P4 TAN EXEC TIME
* P5 DEVICE ID
* P6 DUMMY

**************************************************************
* MODEL COMPONENTS
**************************************************************
* BUSES: GSJS, LBS1...
* CACHES: D11,...,D15
* LEVEL CONTROL: K1, K2
* RED PROC: R2
* DEVICES: D21
* STORAGE: RI, RO
* STORAGE: SI, SO
* STORAGE: SI, TO
* STORAGE: AI, AO
* STORAGE: DI, DO

**************************************************************
* MODEL PARAMETERS
**************************************************************
FILE: CCP         VS1JOB    P4          CONVERSATIONAL MONITOR SYSTEM

INITIAL       X$MODEL,502   MODEL CC/PTB/P
INITIAL       X$VAR,10      DEGREE OF MULTIPROG PER CPU
INITIAL       X$VAR,700    % READ PEQ
INITIAL       X$VAR,900    CONDITIONAL PROB OF FINDING DATA IN L1
INITIAL       X$VAR,1000   IN L2
INITIAL       X$VAR,2       DEVICE SERVICE TIME IN L1
INITIAL       X$VAR,4      IN L2
INITIAL       X$VAR,1,1    BUS SERVICE TIME FOR 8 BYTE BLOCK
INITIAL       X$VAR,8,8    BUS SERVICE TIME FOR 64 BYTE BLOCK
INITIAL       X$VAR,4      DIRECTORY LOCK UP TIME
INITIAL       X$VAR,8      CONTROLLER SERV TIME
INITIAL       X$VAR,1,6    LOOKUP PLUS READ/WRITE TIME OF D11
INITIAL       X$VAR,10000  SIMULATION TIME (IN 10 MS UNITS)
INITIAL       X$VAR,15    TIME R1 SEARCHS DIREC & UPDT LRU STATUS

..........................

- SAVEVALUES
- 
- NTXT TOTAL TXN PROC.
- SUMX TOTAL EXEC TIMES
- SUMX TOTAL WAIT TIMES
- SUMX TOTAL ELAPSED TIM
- 

..........................

- VARIABLES
- 

WRESP F/VARIABLE (X$SUMT/X$NTTXN)  MEAN RESP TIME
TXNT VARIABLE P3-P2    TXN ELAPSED TIME
TXNW VARIABLE P3-P2-P4 TXN WAIT TIME
TXNS VARIABLE P4      TXN EXEC TIME

..........................

- TABLES
- 

TXNT TABLE         VSTXNT,100,100,100
TXNW TABLE         VSTXNW,100,100,100
TXNX TABLE         VSTXNX,100,100,100

..........................

- FUNCTIONS
- 

WHST FUNCTION P1.05
FILE: CCP  VS1:JOB  A4  CONVERSATIONAL MONITOR SYSTEM

1,STAR1/2,STAR2/3,STAR2/4,STAR4/5,STAR5

LOAD FUNCTION  PS.05
1,WW11/2,WW12/3,WW13/4,WW14/5,WW15

LOAD FUNCTION  PS.05
1,WDD11/2,WDD12/3,WDD13/4,WDD14/5,WDD15

LOAD FUNCTION  PS.05
1,RRD11/2,RRD12/3,RRD13/4,RRD14/5,RRD15

LOAD FUNCTION  RN.05
.2,1/.4,2/.6,3/.8,4/1.5

RESSED

* STORAGE FOR L(1) *
* CACHES *

* STORAGE *
S$R:ID11,10/S$SID11,10/S$TID11,10/S$AIR11,10
S$R:ID12,10/S$SID12,10/S$TID12,10/S$AIR12,10
S$R:ID13,10/S$SID13,10/S$TID13,10/S$AIR13,10
S$R:ID14,10/S$SID14,10/S$TID14,10/S$AIR14,10
S$R:ID15,10/S$SID15,10/S$TID15,10/S$AIR15,10

* STORAGE FOR DEVICES *

* STORAGE *
S$R:ID21,10/S$SID21,10/S$TID21,10

* STORAGE FOR REQ PROC *

* STORAGE *
S$3R1:R1,10/S$3R1:R1,10/S$3R1:R1,10/S$3R1:R1,10/S$3R1:R1,10
S$3R2:R2,10/S$3R2:R2,10/S$3R2:R2,10/S$3R2:R2,10

* STORAGE FOR K1 *

* STORAGE *
S$RCK1:K1,10/S$5OK1,10/S$T1K1,10/S$AIK1,10/S$5OK1,10

* STORAGE FOR K2,K3,K4 *
FILE: CCP V51JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 001

 STORAGE $S$R$K2.10/$$S$S$K2.10/$$S$$T1$$K2.10/$$S$$T1$$K2.10/$$S$$D1$$K2.10
 STORAGE $S$S$R$K2.10/$$S$S$S$K2.10/$$S$$T0$$K2.10/$$S$$S$$O$$K2.10/$$S$$O$$D$$K2.10

*******************************
* BOOLEAN VARIABLES *
*******************************

*******************************
* BV FOR READ-THROUGH *
*******************************

*******************************
* BV FOR L(1) *
*******************************

*******************************
* CRR1 BVARIABLE FNUSBUS1*SNF$SR1R1
RDR1 BVARIABLE FNUSBUS1*SNF$SR1D11
RDR12 BVARIABLE FNUSBUS1*SNF$SR1D12
RDR13 BVARIABLE FNUSBUS1*SNF$SR1D13
RDR14 BVARIABLE FNUSBUS1*SNF$SR1D14
RDR15 BVARIABLE FNUSBUS1*SNF$SR1D15
RDS11 BVARIABLE FNUSBUS1*SNF$SID11
RDS12 BVARIABLE FNUSBUS1*SNF$SID12
RDS13 BVARIABLE FNUSBUS1*SNF$SID13
RDS14 BVARIABLE FNUSBUS1*SNF$SID14
RDS15 BVARIABLE FNUSBUS1*SNF$SID15
KRT1 BVARIABLE FNUSBUS1*SNF$TIR1
KRA1 BVARIABLE FNUSBUS1*SNF$AIR1

OKR1 BVARIABLE FNUSBUS1*SNF$ROK1
OKS1 BVARIABLE FNUSBUS1*SNF$SOK1
OKO1 BVARIABLE FNUSBUS1*SNF$OOK1
KDT11 BVARIABLE FNUSBUS1*SNF$TID11
KDT12 BVARIABLE FNUSBUS1*SNF$TID12
KDT13 BVARIABLE FNUSBUS1*SNF$TID13
KDT14 BVARIABLE FNUSBUS1*SNF$TID14
KDT15 BVARIABLE FNUSBUS1*SNF$TID15
KDA11 BVARIABLE FNUSBUS1*SNF$SAID11
KDA12 BVARIABLE FNUSBUS1*SNF$SAID12
KDA13 BVARIABLE FNUSBUS1*SNF$SAID13
KDA14 BVARIABLE FNUSBUS1*SNF$SAID14
KDA15 BVARIABLE FNUSBUS1*SNF$SAID15

*******************************
* BV FOR INTER LEVEL COM*
*******************************
FILE: CCP  VS+JOB  A4  CONVERSATIONAL MONITOR SYSTEM  PAGE 005

| KRR12  BVARIABLE | FNS'LBUS*+SNFSR1K2 |
| KRS12  BVARIABLE | FNS'LBUS*+SNFSR1K2 |
| KKT12  BVARIABLE | FNS'LBUS*+SNFSOK2 |
| KKT21  BVARIABLE | FNS'LBUS*+SNFSOK1 |

| KRR2  BVARIABLE | FNS'LBU52*+SNFSR1R2 |
| KRS2  BVARIABLE | FNS'LBU52*+SNFSR1R2 |
| KRT2  BVARIABLE | FNS'LBU52*+SNFSR1R2 |
| KRA2  BVARIABLE | FNS'LBU52*+SNFSAIK2 |
| KRG2  BVARIABLE | FNS'LBU52*+SNFSAIK2 |
| RDR21  BVARIABLE | FNS'LBU52*+SNFSID21 |
| RDS21  BVARIABLE | FNS'LBU52*+SNFSID21 |
| DK2  BVARIABLE | FNS'LBU52*+SNFSOK2 |
| DNT2  BVARIABLE | FNS'LBU52*+SNFSOK2 |
| DKA2  BVARIABLE | FNS'LBU52*+SNFSAIK2 |
| RK2  BVARIABLE | FNS'LBU52*+SNFSAIK2 |
| RKO2  BVARIABLE | FNS'LBU52*+SNFSAO2 |
| RKA2  BVARIABLE | FNS'LBU52*+SNFSAO2 |

| MACROS |

| MACRO - USE |
| #A FACILITY |
| #A USAGE TIME |

USE STARTMACRO
SEIZE #A
ADVANCE #B
ASSIGN #X,#B
RELEASE #A
ENDMACRO

| MACRO - SEND |
#A FROM
#B TO
#C VIA
#D TRANSIT TIME
#E BY FOR SEND OP

***********************

SEND STARTMACRO
TEST E #E.1
ENTER #B
SEIZE #C
ADVANCE #C
ASSIGN #A.
RELEASE #C
LEAVE #A
ENDMACRO

***********************

MACRO - SND

***********************

SEND STARTMACRO
TEST E #D.1
ENTER #A
SEIZE #B
ADVANCE #C
ASSIGN #C.
RELEASE #B
ENDMACRO

***********************

MACRO - FINI

***********************

FINI STARTMACRO
MARK 3
SAVEVALUE NTXN+1
SAVEVALUE $MPX+.VSTANX
SAVEVALUE $MPY+.VSTANX
SAVEVALUE $MPZ+.VSTANX
SAVEVALUE +RESP,VMRESP
ASSIGN 2.0
ASSIGN 3.0
ASSIGN 4.0
ENDMACRO

***********************

BEGIN SIMULATION

***********************
FILE: CCP VS JOB A4
CONVERSATIONAL MONITOR SYSTEM

SIMULATE

CPU1
PRIORITY 9
MARK 2
ASSIGN 1.1
TRANSFER .X$READ,WWW1,READ

CPU2
PRIORITY 9
MARK 2
ASSIGN 1.2
TRANSFER .X$READ,WWW1,READ

CPU3
PRIORITY 9
MARK 2
ASSIGN 1.3
TRANSFER .X$READ,WWW1,READ

CPU4
PRIORITY 9
MARK 2
ASSIGN 1.4
TRANSFER .X$READ,WWW1,READ
FILE: CCP VS1JOB A4 CONVERSATIONAL MONITOR SYSTEM

- CPU #5

-----------------------------------
CPU5  GENERATE     ...,$MAXMP...F
STARS  PRIORITY  9   SET HIGH P FOR NEW TXN
         MARK  2      ARRIVAL TIME
         ASSIGN  1.5  CPU ID
         TRANSFER .XENREAD,WWW1,READ

-----------------------------------
DATA TO BE READ

-----------------------------------
READ  ASSIGN  11.0
         ASSIGN  5,FN$LOAD
         TRANSFER ,FN$RUNIT

-----------------------------------
DEVICE D11

-----------------------------------
RDO11  ASSIGN  5.1
         TRANSFER ,XSPIN1,NIN11,RIN11
RIN11  ASSIGN  11.0
        SND  MACRO  RDO11,LIBU1,X$BEX1,BV$RDR11
        USE  MACRO  DRO11,X$DEX1
        SEIZE  LIBU1
        ADVANCE  X$BEX1
        ASSIGN  4*,X$BEX1
        RELEASE  LIBU1
        LEAVE  RDO11
        FINI  MACRO  SPLIT  1,FN$WCHST
        TERMINATE
NIN11  ASSIGN  11.0
         PRIORITY  0
        SND  MACRO  RO11,LIBU1,X$BEX1,BV$DKR1
         TRANSFER ,COMR

-----------------------------------
DEVICE D12

-----------------------------------
RDO12  ASSIGN  5.2
TRANSFER .AS$X4,11N12,RIN12
RIN12 ASSIGN 11.0
SND MACRO R1D12,LBUS1,X$BEX1,BV$RD12
USE MACRO DQP12,X$DE1
SEIZE LB$51
ADVANCE X$BEX1
ASSIGN 4-X$BEX1
RELEASE LB$51
LEAVE R0D12
FINI MACRO SPLIT 1.FNS4CHST
TERMINATE
RIN12 ASSIGN 11.0
PRIORITY 0
SND MACRO R1C11,LBUS1,X$BEX1,BV$DKR1
TRANSFER .CMXR

-----------------------------
* DEVICE D13 *
* *
-----------------------------
R1D13 ASSIGN 5.3
TRANSFER .AX$241,11N13,RIN13
RIN13 ASSIGN 11.0
SND MACRO R1C13,LBUS1,X$BEX1,BV$RD13
USE MACRO DQP13,X$DE1
SEIZE LB$51
ADVANCE X$BEX1
ASSIGN 4-X$BEX1
RELEASE LB$51
LEAVE R1D13
FINI MACRO SPLIT 1.FNS4CHST
TERMINATE
RIN13 ASSIGN 11.0
PRIORITY 0
SND MACRO R1C11,LBUS1,X$BEX1,BV$DKR1
TRANSFER .CMXR

-----------------------------
* *
* DEVICE D14 *
* *
-----------------------------
R1D14 ASSIGN 5.4
TRANSFER .AX$241,11N14,RIN14
RIN14 ASSIGN 11.0
SND MACRO R1D14,LBUS1,X$BEX1,BV$RD14
USE MACRO DQP14,X$DE1
SEIZE LB$51
FILE: CCP

VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 010

ADVANCE X$BEX1
ASSIGN 4+, X$BEX1
RELEASE LBUS1
LEAVE RID14

FINI MACRO
SPLIT 1,FNSWCHST
TERMINATE

NIN14 ASSIGN 11,0
PRIORITY 0

SND MACRO R0K1, LBUS1, X$BEX1, BV$DKR1
TRANSFER .COMR

**********************************************************************

* DEVICE D15

**********************************************************************

R0D15 ASSIGN 5,5
TRANSFER .XSPIN1, NIN15, RIN15
RIN15 ASSIGN 11,0

SND MACRO RID15, LBUS1, X$BEX1, BV$RDR15
USE MACRO DRP15, X$DEX1
SEIZE LBUS1
ADVANCE X$BEX1
ASSIGN 4+, X$BEX1
RELEASE LBUS1
LEAVE RID15

FINI MACRO
SPLIT 1,FNSWCHST
TERMINATE

NIN15 ASSIGN 11,0
PRIORITY 0

SND MACRO R0K1, LBUS1, X$BEX1, BV$DKR1
TRANSFER .COMR

**********************************************************************

* DATA TO BE WRITTEN IS IN LEVEL 1

**********************************************************************

W0W1 ASSIGN 11,0
ASSIGN 5,FNSLOAD
TRANSFER .FNSWUNIT

**********************************************************************

* DEVICE D11

**********************************************************************
FILE: CCP VS1JOB A4

CONVERSATIONAL MONITOR SYSTEM

PAGE 011

******************************************************************************

5.1
ASSIGN WDO11
MACRO SID11, LBUS1, X$BEX1, BV$RDS11
USE MACRO DRP11, X$DEX1
PRIORIY 0
SEND MACRO SID11, SOK1, LBUS1, X$BEX8, BV$SDK1
FINI MACRO
SPLIT 1, COMW
TERMINATE
SPLIT 1, FN$WCHST

******************************************************************************

5.2
ASSIGN WDO12
MACRO SID12, LBUS1, X$BEX1, BV$RDS12
USE MACRO DRP12, X$DEX1
PRIORIY 0
SEND MACRO SID12, SOK1, LBUS1, X$BEX8, BV$SDK1
FINI MACRO
SPLIT 1, COMW
TERMINATE
SPLIT 1, FN$WCHST

******************************************************************************

5.3
ASSIGN WDO13
MACRO SID13, LBUS1, X$BEX1, BV$RDS13
USE MACRO DRP13, X$DEX1
PRIORIY 0
SEND MACRO SID13, SOK1, LBUS1, X$BEX8, BV$SDK1
FINI MACRO
SPLIT 1, COMW
TERMINATE
SPLIT 1, FN$WCHST

******************************************************************************

5.4
ASSIGN WDO14
MACRO SID14, LBUS1, X$BEX1, BV$RDS14
USE MACRO DRP14, X$DEX1
PRIORIY 0
SEND MACRO SID14, SOK1, LBUS1, X$BEX8, BV$SDK1
FINI MACRO
SPLIT 1, COMW
TERMINATE
SPLIT 1, FN$WCHST

******************************************************************************
FILE: CCP VS1JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 012

WDD14 ASSIGN 5.4
SEND MACRO SID14, LBUS1, X$BE1, 8V$RD514
USE MACRO DNP14, X$DE1
PRIORIT Y 0
SEND MACRO SID14, SOK1, LBUS1, X$BE8, 8V$DK51
SPLIT 1, CONW
FINI MACRO
SPLIT 1, FNSWCHST
TERMINATE

WDD15 ASSIGN 5.5
SEND MACRO SID15, LBUS1, X$BE1, 8V$RD515
USE MACRO DNP15, X$DE1
PRIORIT Y 0
SEND MACRO SID15, SOK1, LBUS1, X$EX8, 8V$DK51
SPLIT 1, CONW
FINI MACRO
SPLIT 1, FNSWCHST
TERMINATE

* COMMON CODE FOR READ REQUEST *

COMM ASSIGN 11.0
USE MACRO KRP1, X$KE1
SEND MACRO ROK1, RIK2, GBUS, X$BE1, 8V$KKR12
USE MACRO KRP2, X$KE1
SEND MACRO RIK2, RIK2, LBUS2, X$BE1, 8V$KKR2
USE MACRO RRP2, X$KE1

* READ DATA IS FOUND IN L(2) *
**DATA IS IN D2**

**REPLACE**

```
RRR21 ASSIGN 1.0
SEND MACRO C:82.10D21, LBUS2, XSBEX1, BV$RDR21
USE MACRO C:82, A$D$E$X2
SEND MACRO E:82, TCK2, LBUS2, XSBEX8, BV$D$K$T2
```

**READ-THROUGH TO D1**

**STORE DATA INTO (.1) AS RESULT OF A READ-THROUGH**

```
STOR1 ASSIGN 11.0
USE MACRO A:81, X$S$E$X
SPLIT 1, FNSWICH
TERMINATE
```

**RT STORE INTO D11**

```
WWW11 ASSIGN 11.0
SEND MACRO TIX1, TID11, LBUS1, XSBEX8, BV$KDT11
USE MACRO C:81, X$D$E$X1
```
FILE: CCP VS1JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 014

SEND WACRO TID11,OOK1,LIBS1,X$SEX1,BV$DKO1
SPLIT 1.0VF11
FINI WACRO
SPLIT 1,FN$WCHST
TERMINATE

0VF11 TRANSFER .0VL1

* RT STORE INTO D12
* *
* *

WWW12 ASSIGN 11.0
SEND WACRO TIK1,TID12,LIBS1,X$SEX8,B.$SKT12
USE WACRO DRP12,X$SEX1

SEND WACRO TID12,OOK1,LIBS1,X$SEX1,BV$DKO1
SPLIT 1.0VF12
FINI WACRO
SPLIT 1,FN$WCHST
TERMINATE

0VF12 TRANSFER .0VL1

* RT STORE INTO D13
* *
* *

WW13 ASSIGN 11.0
SEND WACRO TIK1,TID13,LIBS1,X$SEX8,B.$SKT13
USE WACRO DRP13,X$SEX1

SEND WACRO TID13,OOK1,LIBS1,X$SEX1,BV$DAD1
SPLIT 1.0VF13
FINI WACRO
SPLIT 1,FN$WCHST
TERMINATE

0VF13 TRANSFER .0VL1

* RT STORE INTO D14
* *
FILE: CCP  VS JOB  A4       CONVERSATIONAL MONITOR SYSTEM       PAGE 015

***************
WWW14 ASSIGN   11,0  
SEND MACRO    TIK1,TID14,LE-1,X$8EX8,BV$KDT14  
USE MACRO    DRP14,X$DEX1  
SEND MACRO    TID14,OK1,LE-1,X$8EX1,BV$DK01  
SPLIT       1.OVF14  
FINI MACRO   1,FS%CHST  
             TERMINATE  
OVF14 TRANSFER    .OV11

***************
WWW15 ASSIGN   11,0  
SEND MACRO    TIK1,TID15,LBUS1,X$8EX8,BV$KDT15  
USE MACRO    DRP15,X$DEX1  
SEND MACRO    TID15,OK1,LBUS1,X$8EX1,BV$DK01  
SPLIT       1.OVF15  
FINI MACRO   1,FH$WCHST  
             TERMINATE  
OVF15 TRANSFER    .OV11

***************
OV11 ASSIGN   11,0  
USE MACRO    KRP1,X$9EX  
SEND MACRO    OK1,OK2,GBUS,X$8EX1,BV$KDO12  
USE MACRO    KRP2,X$9EX  
SEND MACRO    OK2,OK2,LBUS2,X$9EX1,BV$KRO2  
USE MACRO    RRP2,X$9EX
FILE: CCP VS1JOB A4 CONVERSATIONAL MONITOR SYSTEM PAGE 016

LEAVE DIR2
TERMINATE

* COMMON CODE FOR STORE-BEHIND

COMM ASSIGN 11,0
USE MACRO KRP1.X$KEX
SEND MACRO SOK1.SIK2.GBUS.X$SBE8.BV$KKS12
USE MACRO KRP2.X$KEX
SEND MACRO SIK2.SIR2.LBUS2.X$SBE8.BV$KRS2
USE MACRO RRP2.X$REX
SEND MACRO SIR2.SID21.L9L52.X$SBE8.BV$SD521
USE MACRO DRP21.X$DEX2
SEND MACRO SID21.AOK2.LBUS2.X$SBE1.BV$SDKS2

ACK FROM L(2) TO L(1)

USE MACRO KRP2.X$KEX
SEND MACRO AOK2.AI1,GBUS.X$SBE1.BV$KKA21
USE MACRO KRP1.X$KEX
LEAVE AI1

TERMINATE

SIMULATION CONTROL