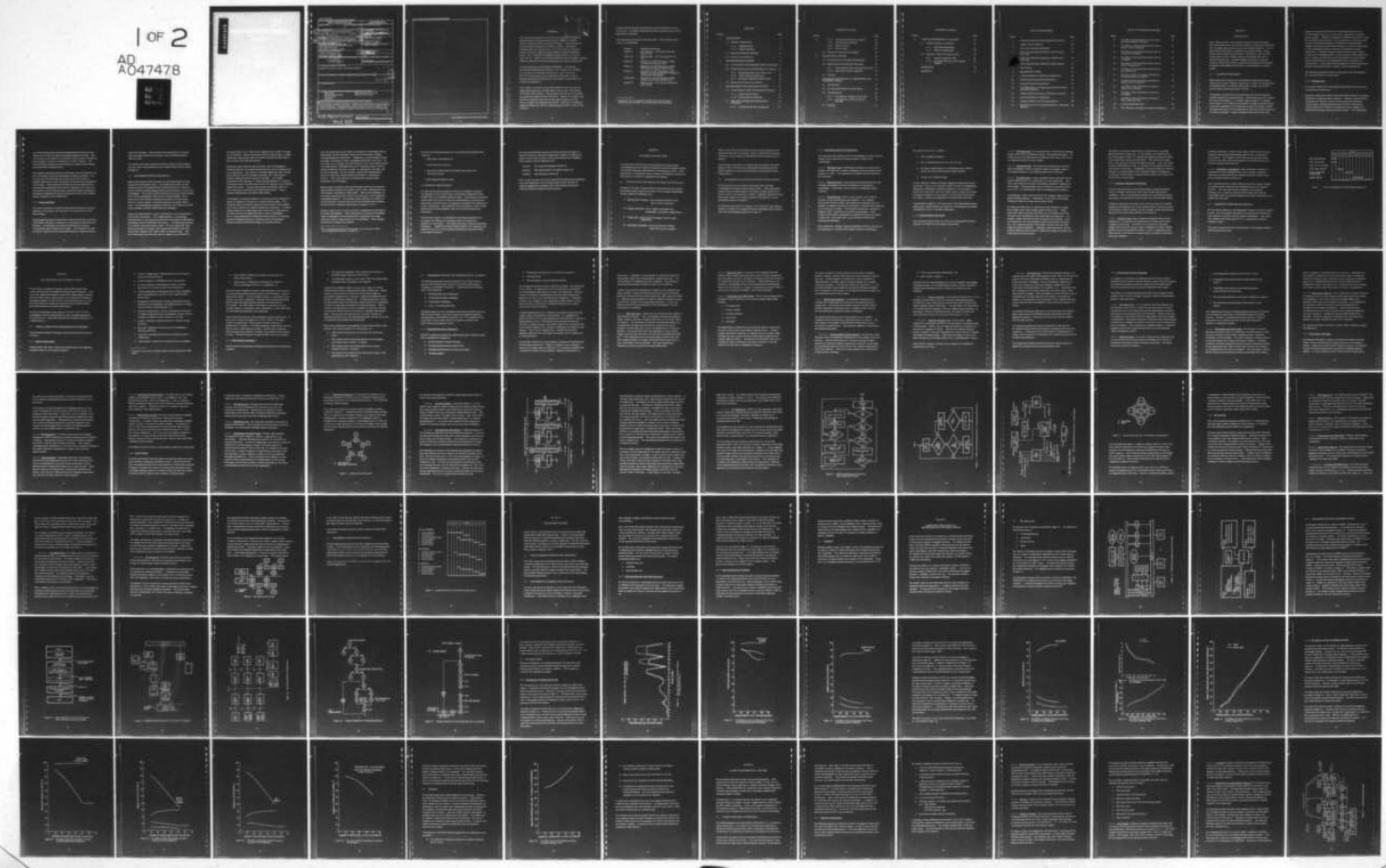
AD-A047 478 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
DISTRIBUTED DATA PROCESSING TECHNOLOGY. VOLUME VII. APPLICATION--ETC(U)
SEP 77 R G ARNOLD, W E BOEBERT, E D JENSEN DASG60-76-C-0087

UNCLASSIFIED 77SRC71 NL

1 OF 2
AD
A047478



AD A047478

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOV'T ACCESSION NUMBER	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (AND SUBTITLE) (6) Distributed Data Processing Technology. Volume VII. Application of DDP Technology to BMD: Experiments.		5. TYPE OF REPORT/PERIOD COVERED (9) Final Report. Oct 1976 - Oct 1977,
		6. PERFORMING ORG. REPORT NUMBER (14) 77SRC71
7. AUTHOR(S) (10) R. G. Arnold, W. E. Boebert, E. D. Jensen, G. D. Marshall, W. K. Utt, R. Y. Kain, R. Wood, R. Stone, D. Palmer		8. CONTRACT OR GRANT NUMBER(S) (15) DASG60-76-C-0087
9. PERFORMING ORGANIZATIONS NAME/ADDRESS Honeywell Systems and Research Center 2600 Ridgway Parkway Minneapolis, Minnesota 55413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME/ADDRESS Ballistic Missile Defense Advanced Technology Center Huntsville, Alabama 35807		12. REPORT DATE (11) Sep 1977 13. NUMBER OF PAGES 122
14. MONITORING AGENCY NAME/ADDRESS (IF DIFFERENT FROM CONT. OFF.) (12) 121p.		15. SECURITY CLASSIFICATION (OF THIS REPORT) Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (OF THIS REPORT)

Approved for public release; distribution unlimited.

DDC RECEIVED DEC 8 1977 F

17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

Experiments
Distributed processing
Architectures
Algorithms
DDP experimental facilities
BMDATC ARC facility

20. ABSTRACT (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

Critical issues from other volumes are reviewed and experiments and case studies are recommended to resolve these issues. Results of one architecture simulation experiment are discussed. An experiment facility for DDP for the ARC at BMDATC and required hardware and software to support the proposed experiments are presented.

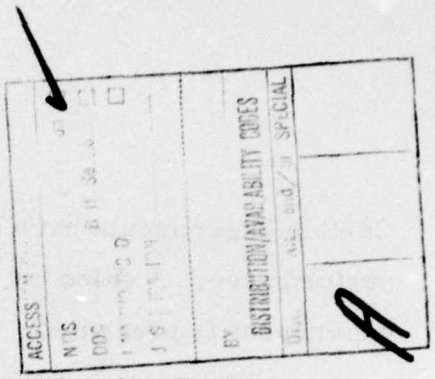
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

402 349

LB



FOREWORD

The research documented in this volume was conducted under Ballistic Missile Defense Technology contract number DASG60-76-C-0087, entitled "Distributed Data Processing Technology." The work was performed by Honeywell Systems and Research Center, Minneapolis, Minnesota under the direction of Mr. C. R. Vick, Director, Data Processing Directorate, Ballistic Missile Defense Advanced Technology Center. Mr. J. Scalf was the BMDATC project engineer for this contract; Ms. B. C. Stewart was the Honeywell/GRC program manager. General Research Corporation acted as subcontractor for a portion of this work.

The experiment plan was developed at Honeywell. The following personnel participated in this research: R. G. Arnold, W. E. Boebert, E. D. Jensen, G. D. Marshall, and W. K. Utt. R. Y. Kain of the University of Minnesota acted as consultant to Honeywell. General Research Corporation's observations on the subject of experiments are presented as Appendix A.

In this report, we review critical issues defined in other volumes and suggest experiments that will help address these issues concerning the use of DDP in BMD systems. We also discuss the results of one simulation experiment conducted under this project. Finally, we propose a configuration of hardware and software that might be added to the ARC system to support the suggested experiments. Reactions to a similarly suggested configuration that was independently generated by SDC are also presented.

Certain experiments have been identified as prime candidates for early performance. A calendar scheduling the phases required to perform each experiment is presented.

This document is Volume VII of the final report. Other volumes of the report are the following:*

Volume I	-	Management Summary
Volume II	-	DDP Rationale: The Program Planning Point of View
Volume III	-	DDP Rationale: The Technology Point of View
Volume IV	-	Application of DDP Technology to BMD: Architectures and Algorithms
Volume V	-	Application of DDP Technology to BMD: DDP Subsystem Design Requirements
Volume VI	-	Application of DDP Technology to BMD: Impact on Current DP Subsystem Design and Development Technologies
Volume VIII	-	Application of DDP Technology to BMD: Research Performance Measurement
Volume IX	-	DDP Rationale: The Program Experience Point of View

---

* Volumes V, VI, the appendix to Volume VII, and a section of Volume VIII were prepared by General Research Corporation.

CONTENTS

CONTENTS (continued)

CONTENTS (concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (concluded)

SECTION 1

INTRODUCTION

Paper studies and even formal analyses (using such tools as queueing theory and graph theory) are ultimately limited in the amount of information they can supply about a system of more than trivial complexity. Because DDP hardware and software architectures are highly complex, design studies must be backed by experiments. Furthermore, since the DDP design process is also complex, studies of the process must also be backed by experiments. Properly designed experiments can greatly facilitate concept comprehension, hypothesis formulation and verification, feasibility or infeasibility demonstration, and quantification of solution payoffs.

1.1 CLASSES OF EXPERIMENTS

Experiments will be proposed to test hypotheses concerning the system design process (called "designer experiments" herein), the implementation mechanisms (called "mechanism experiments" herein), and the DDP payoffs (called "payoff experiments" herein).

The purpose of an experiment is an important factor in determining the benefits expected from its performance. Most experiments can be divided into one of three classes: 1) to verify a logical design, 2) to quantify performance, or 3) to select a best technique or design among alternatives. A gate-level simulation of a new processor design, for example, verifies the logical correctness of the mechanism but would not be used to determine the system throughput. Logical verification experiments range from

complete correctness verification to concept familiarization in which system designers become more aware of the consequences of possible design decisions. Similarly, performance verification experiments range from feasibility demonstrations, which ascertain that a particular design concept (a mechanism experiment) or a particular design process (a designer experiment) is not terribly inefficient, to success measurements in which detailed performance measures are determined (a payoff experiment). Alternative selection experiments also range along the continuum from feasibility studies to success measurements, though they may be of little use when they provide little confidence in their results. General experiments are most appropriate for the early phases of a design project. At this stage of understanding of the DDP design process, designer experiments should be confined to general experiments.

The following subsections discuss two important factors contributing to the costs and benefits of an experiment.

1.1.1 Modeling Detail

The amount of detail in the experimental model affects the time and cost in performing an experiment.

Mechanism experiments with the least detail include functional simulations performed on a general-purpose computer. Typically, these functional simulations use stochastic models to determine gross performance characteristics, including queueing delays and sites of bottlenecks. Further detail can be provided by emulating the system design at the instruction level using a general-purpose computer. Such experiments

can provide limited information concerning performance (because many steps must be emulated to obtain statistically significant results). More accurate data can be obtained by emulating the system using a collection of general-purpose computers coordinated by a master computer. The ultimate detail is provided by implementing the complete system (though not necessarily in deployable form).

Most designer experiments will have little detail, since the problems will have to be simple in order that unambiguous results be produced. It is not clear how one could effectively increase the amount of detail in designer experiments because of problems with the statistical significance of the results. Early experiments should use the least possible detail, not only because additional detail would not contribute to the results, but also because it may be dependent upon a particular design methodology or mechanism technology that cannot be projected to the time when complete designs or systems may be constructed.

1.1.2 System Specificity

The system specificity of a proposed experiment determines the scope of application of the results, and, therefore, the benefits for the overall system design.

Broadly applicable (generic) experiments study characteristics common to many designs, functions, architectures, or pairings of architectures with functions. An experiment to measure the performance of various communication policies falls in this category. At the opposite end of the spectrum lie experiments that ascertain the behavior of specific function-

architecture pairings. Measurements of the performance of a specific radar scheduling architecture executing a radar scheduling algorithm typify this class.

The majority of the first experiments should be generic so that designers may gain a better understanding of system behavior under various design philosophies.

1.2 EXPERIMENT SELECTION RATIONALE

Experiments should be selected because the anticipated benefits are high compared to the anticipated costs. The benefits must include not only benefits in terms of the payoffs of DDP to BMD (discussed in Volume II), but also benefits in understanding how DDP systems should be characterized and designed. These latter benefits include such items as an understanding of how design methodologies might bias the resultant designs, of how system parameters influence computation demands, and of how logical structure must be imposed on distributed data bases to ensure correct usage. Such understanding contributes indirectly to the efficacy of all DDP systems designs, and thus to BMD systems in particular.

Another important benefit of a series of experiments can be determination of parameter sensitivities. As we design systems, we make design decisions based on our estimates of parameters describing the technological possibilities and the environment of the system. We really should estimate our confidence in these parameter values. For every experiment which uses the parameter as an input, other experiments should be made with the parameter taking on other values within its estimated range. This set of experiments determines the effect of changes in the parameter on

the experimental results, which is the sensitivity of the results to changes in the parameter. When a parameter's effects are quite large, we should design other experiments to more accurately estimate the actual value of the parameter in the BMD environment.

Experiment costs include not only the obvious costs of developing the hardware/software for the experiment but also the costs of meeting the run-time requirements. The search for statistical significance from complex experiments can lead one to propose unrealistic experiments unless a properly configured hardware system is available. The complex asynchronism and concurrency, coupled with unpredictable (i.e. random) interactions between the BMD system and the DP subsystem make statistical significance elusive. One may easily propose experiments whose run-times would exceed the MTBF of the experimental facility! Proper hardware configurations can alleviate this problem.

The experiment complexity problem is very serious when the experiment involves simulation or emulation of a complete system design. This complexity affects not only the significance of the results, but also dollar cost and time required to develop and perform the experiment. Complete design emulation is an inappropriate type of experiment at this phase of our efforts towards developing DDP system design methodologies. First, we must experiment to quantify the characteristic of the essential features of distributed systems, and thereby show that distributed systems do exhibit BMD payoffs lacking in centralized systems.

The cross-interaction matrix (Volume II) showing the relationships between BMD payoffs and DDP attributes would seem the natural place to begin selecting appropriate experiments. Though this is a good strategy, present knowledge is inadequate to define meaningful experiments at this level. We do not know how to quantify many DDP attributes, and we also cannot quantify many BMD payoffs.* Meaningful experiments emulating complete systems could be performed, but these, as noted, are too specific, too costly, and too time-consuming. We can, however, specify a few experiments to learn more about some payoff-attribute interactions, as discussed in Section 4. Much more research is required before specific payoff experiments can be developed.

Specific system mechanism experiments should be performed to study the implementation techniques that could be used to endow a distributed system with attributes contributing to BMD payoffs. With current knowledge, we can propose experiments that evaluate the performance of these properties of distributed systems. These implementation problems are subjects of current research; several of them do not have well-understood solutions. We discuss these mechanism experiments in Sections 2.3 and 3.

Designer experiments are needed to study the consequences and benefits of the design methodologies. These experiments are selected by examining design methodologies and determining areas requiring further design attempts to resolve critical issues in the methodologies. These designer experiments are discussed in Section 2.

---

*For a complete discussion of this problem, see the section entitled Toward Quantifiable Payoffs in Volume II.

We answer the following questions for each experiment described in this document:

1. Why perform the experiment?

2. What results are expected?

3. How do the results relate to the BMD means-ends cross-interaction matrix?

4. What approach should be used?

1.3 SURVEY OF THIS DOCUMENT

The experiments planned during this contract are designed to assist in the resolution of some critical issues concerning the application of DDP to BMD which were identified during this study. This document summarizes critical issues that may be resolved by experimentation, describes some experiments, prioritizes them, and proposes schedules for the performance of these experiments. Some tools that must be developed to assist certain experiments have been identified; these are described along with the experiments.

As part of this contract, we did perform one simulation experiment to demonstrate the adequacy of simulation models and to gain some experience in such studies. This experiment and its results are described in Section 5. Suggestion for experimental facilities were developed, and Honeywell reacted to a facility proposal independently generated by SDC, as described in Section 6.

The major content of this document comprises Sections 2 through 4 in which we present the experiment suggestions, organized according to the volumes of this report in which relevant critical issues were identified. These sections cover the following areas:

Section 2: DP Design Methodologies (Volume IV)

Section 3: DDP Implementation Techniques (Volume IV)

Section 4: DDP Rationales (Volume II)

There is some variance in the level of detail in the experiment descriptions; some early suggestions were developed in great detail, but later we collected suggestions and included them in this document to provide a record of all ideas developed during this project.

SECTION 2

DDP DESIGN METHODOLOGIES

We review some critical issues concerning data processing design methodologies that were uncovered during this project and propose designer experiments to discover additional facts concerning these issues. Some experiments are predicated on prior completion of additional research and development, as noted in the appropriate descriptions.

2.1 CRITICAL ISSUES CONCERNING DDP DESIGN METHODOLOGIES

In Volume IV, Section 7 (Future Needs) we described numerous critical issues that should be addressed in further work on design methodologies. The list included the following areas:

1) Specification Techniques--how to specify properties of the system and its components.

2) Design Constraints--how to confine the designer from unreasonable, unrealistic configurations.

3) Design Steps--how to direct the designer towards a good configuration.

4) Description Techniques--how to describe the complete system that has been designed.

Many of these issues require further research and development before experimentation will be worthwhile; others, including selected design constraints, can be subjects of experiments in the immediate future, if desired.

In this section we describe a few experiments that might be performed during FY 78 or FY 79 and some that should be performed only after further progress in research and development concerning the related critical issues.

2.2 EXPERIMENTS CONCERNING DDP DESIGN METHODOLOGIES

We next describe several designer experiments that test design methodologies and suggest improvements in them. These experiments require the use of actual designers who perform design exercises and then are queried concerning their methods. Additional evaluations can be based on the quality of the designs produced (assuming that we have a quantifiable measure of "quality").

The following experiments are designed to test hypotheses concerning the viability of an enforced or suggested design methodology and the modifications to the methodologies proposed in Volume IV.

2.2.1 Methodology Enforcement Experiment

This experiment will evaluate the use of methodologies for DDP subsystem designs and the importance of forcing designers to follow those methodologies.

2.2.1.1 Why Experiment?--Methodology development is useful only if designers will follow the resulting methodology and if they produce better designs as a result. This experiment is intended to provide answers to these questions.

2.2.1.2 Expected Results--This experiment should document the value of design methodologies and uncover difficulties in enforcing the methodology. Furthermore, ways to enforce methodologies will be compared.

2.2.1.3 The Experiment--First we must develop a way to enforce particular design methodologies upon designers. One candidate enforcement technique requires documentation of all design steps; the designer is required to describe in writing how he followed each design step as he performed the design. In the proposed experiment, one group of designers will be required to use specified methodologies. Other designers will be given methodology descriptions but will not be required to follow them, and a final control group will have no information regarding methodologies.

In the experiment, designers using methodologies available at the time of the experiment are compared with designers using no methodology.

The experiment proceeds as follows:

1. Select a group of designers.

2. Select a BMD function to serve as the test case.

3. Randomly assign designers to methodologies and to whether and how they must document their design decisions.

4. Compare the resulting designs.

It will be difficult to obtain statistically significant results from such an experiment; however, it is important to know the value of methodologies. Quantitative architectural evaluation techniques are not essential to perform this experiment, since the designs could be compared by a panel of designers (though the same designers may have biases towards their own designs, and thus are inappropriate to evaluate their own work).

A questionnaire designed to elicit information concerning design decisions might produce helpful insights concerning new methodological possibilities and improvements to the existing methodologies.

2.2.2 Payoff Emphasis Experiment

This experiment will determine the effects of presenting DDP subsystem designers with different payoff emphasis information.

2.2.2.1 Why Experiment?—It is important that we determine how strongly designers are affected by differing emphases concerning design goals. We cannot ascertain such information by nonexperimental means, since no one can explain the behavior of human beings.

2.2.2.2 Expected Results--This experiment should improve our knowledge concerning the effects of emphasis changes on designer behavior. New design techniques to meet different goals may be uncovered.

2.2.2.3 The Experiment--A team of designers will be presented with the identical BMD function to be implemented in a DDP architecture. They will be given different directions concerning the payoff(s) to emphasize in their designs. Resulting designs will be evaluated (as described in Subsection 2.2.1.3) to determine the effects of changing payoff emphases.

A questionnaire could be used to learn how the designers approached the different design problems. Additionally, we may collect novel design techniques useful for meeting various payoff criteria.

In a follow-on experiment, we would ask the same designers to modify their designs to also achieve higher payoffs in other areas. Useful information might be gained by selecting, say, two payoffs for the first experiment and reversing their roles in the follow-on experiment. For example, suppose that fault detection and growth were selected. In the first experiment, one set of designers would be told that fault detection was a very important property of the design, while the other set would design for growth possibilities. During the second experiment, the first group of designers would be asked to modify their designs for growth, the second group for fault detection.

The follow-on experiment could produce several outcomes, depending upon architectural comparisons. If, for example, the modifications in the second stage are minor, we conclude that designs for growth are likely to have good fault-detection characteristics. This information belongs in the cross-interaction matrix describing the payoffs. On the other hand, if major modifications are made, we consider the two payoffs unrelated or even counter to each other. Additionally, if the final designs are radically different, we conclude that the order of payoff handling was significant. A selection of which should be emphasized first requires more study of the overall system requirements.

2.2.3 Component Degradation Experiments

We need to learn good ways of specifying degraded component behavior. These two designer experiments will evaluate possible specification techniques. Since no specification techniques have been proposed during this project, this experiment cannot be performed in the immediate future.

Though the experiments described in this section are based on component degradation specifications, the reader should feel free to construct similar experiments to study the impacts and true meanings of other specifications discussed under Future Needs (Section 7) in Volume IV.

2.2.3.1 Designer Usage--Since component degradation affects system performance, we must place this experiment in a context where system reliability is paramount. This effect can be achieved by instructing designers that they shall consider system availability (or another similar payoff) to be the most important attribute. Cost or configuration limitations will have to be imposed to exclude arbitrary redundancy and achieve ultra-high reliability.

During the experiment, designers will be given a BMD function and some component specifications incorporating selected forms of degradation specifications. The designers will be asked to document their design procedures and will be given questionnaires to determine how they have used the degradation specifications.

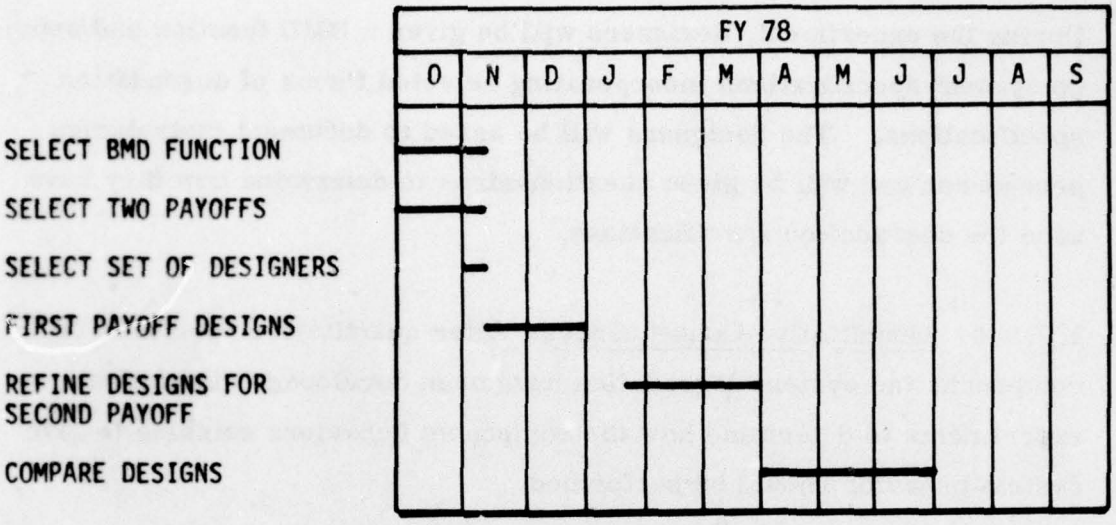
2.2.3.2 Quantitative Consequences--After quantitative measures of component and system degradation have been developed, simulation experiments to determine how the component behaviors combine to give system behavior should be performed.

This combination issue is closely related to the system control strategies concerning resource allocation and scheduling in the face of degraded behavior. The experiment should be designed so that alternate strategies can be compared in specific BMD contexts and so that the relationships between component degradation and system degradation can be studied more generally.

2.3 EXPERIMENT PRIORITIES AND SCHEDULE

We feel that the experiment concerning payoff emphases should have highest priority, since its potential consequences are important for development and refinement of computer systems design methodologies. This experiment should be performed during FY 78; an approximate schedule is outlined in Figure 1.

The other two experiments have lower priorities; they probably should be deferred until future years.



Figu[illegible]. Proposed Schedule for Payoff Emphasis Experiment

SECTION 3

DDP IMPLEMENTATION TECHNIQUE STUDIES

In this section, we propose mechanism experiments to help resolve certain critical issues concerning DDP implementation techniques. Most, but not all, experiments proposed to study DDP implementation techniques will cover either system control policies or interconnection mechanisms. Although the policies are affected by the interconnections and vice-versa, these interactions are usually less important than the separate effects that will be studied by the proposed experiments.

We review the identified critical issues (in Section 3.1) and note their relationship to current research before we turn to detailed experiments (in Subsection 3.2). Experiment priorities and schedules (Subsection 3.3) complete Section 4.

3.1 CRITICAL ISSUES IN DDP IMPLEMENTATION TECHNIQUES

Implementation techniques encompass system control and interconnection techniques.

3.1.1 System Control Issues

Critical issues in the system control area include issues in the following categories (which are not mutually exclusive):

- Process* configuration--constructing processes from pieces of user and system software.
- Process assignment--binding processes to processors.
- Process scheduling--determining the sequence in which processes are to execute on one or more processors.
- Process dispatching--activating processes in their scheduled order and mapping them into one or more control streams (processors).
- Interprocess communication--the flow of information among processes for such purposes as providing operands, accessing shared variables, synchronization, etc.
- Process synchronization--the time coordination of processes to maintain proper precedence relationships, to share common variables, to allocate common resources, and so on.
- Resource allocation--passing control over resources among processes.
- Data base management--performance and coordination of data base operations.
- Protection--guarding information from unauthorized use or modification.
- Fault detection--discovery of erroneous events or conditions.

---

*"Process" is used in its computer science sense rather than its BMD sense.

- Fault isolation--confinement of faults and their effects to a subset of the system.
- Fault recovery--tolerating or overcoming the effects of a fault by degrading performance, reconfiguring, etc.

For each of these categories of system control functions, much remains to be learned about alternate concepts, policies, and mechanisms, even in a uniprocessor context. For the concepts, policies, or mechanisms of any category, the issues, alternatives, and trade-offs are poorly understood. System design and implementation decisions are almost all made heuristically. In a DDP context, the situation is even worse: the few concepts, policies, mechanisms, issues, alternatives, or trade-offs which are well understood stand out as rare exceptions.

The current state of the art varies among these system control issues. In many areas, policies that implement "correct" behavior are known. Usually, the performance of these policies cannot be analyzed, so experiments are required. Very little information concerning the overhead incurred under various policies is known. In other areas, inefficient policies giving correct behavior are known, but no efficient correct policy is known. Distributed data base management falls in the latter category.

3.1.2 Interconnection Techniques

Critical issues in interconnection techniques include issues in the following categories:

- Interconnection topologies--What should be the structure of the paths among elements of a DDP system?
- Communication medium access control--How do communications protocols resolve competing access requests?

A shared data transport medium (such as a bus, a loop, or a shared memory) occurs in many existing and forseeable architectures. Shared memory schemes have been treated extensively in the analysis of conventional multiprocessors, both from a logical synchronization view[1] and from a performance analysis view. To study performance, one can develop analytic models[2] which provide results limited by the assumptions necessary to make analysis possible or develop simulations. Shared memory communications may not be acceptable in many other real-time command and control systems, since they are a possible central failure point. Some bus and loop control schemes have been modeled and simulated (references are in the following discussion), but not in real-time command and control application environments.

Some of the critical issues concerning bus or loop schemes which require answers, and are thus candidates for experiments, are:

- Under what circumstances are buses better (or worse) than loops?
- What control scheme and protocol offers the best throughput and response time for buses? for loops?
- What control scheme and protocol offers better systems integrity for buses? for loops?
- How should the bus (loop) interface unit hardware appear to the application/executive software?

3.2 EXPERIMENTS STUDYING DDP IMPLEMENTATION TECHNIQUES

The following subsections describe independent sets of experiments to improve our knowledge of the critical issues in DDP implementation techniques, as described in Section 3.1. Experiments in the following areas will be discussed:

- Distributed data base management
- Decentralized resource allocation
- Decentralized scheduling
- Loop vs. bus interconnections

During this project we ran a simulation to determine the performance of a shared memory within a real-time command and control environment: this experiment is described in Subsection 3.2.4. The remainder of the experiments covered in this volume are proposals; no efforts have begun towards performing any of them.

3.2.1 Distributed Data Base Management

Data base management includes the following operations, whether the data base is centralized or distributed:

- Performing data accessing strategies
- Interfacing with processes using the data
- Checking authorizations for all access attempts
- Validating updates

- Monitoring the performance of the data base subsystem
- Detecting errors
- Recovering from errors (possibly backing up)

We can divide the data base system design into two parts: data storage and access control. In uniprocessor systems, both parts are centralized; the corresponding design problems are quite well understood. When the control is centralized, straightforward extensions of the uniprocessor policies can be used, even if the data itself is distributed. Centralized control is not acceptable in many applications of DDP to BMD, however, since the central control point is a potential bottleneck and a critical element with respect to failures.

A simple distributed data base technique is to partition the data base and assign a single control point for each partition. If a multiplicity of processors need access to any particular part of the data base, a number of difficult problems arise; these include how to partition the data base, and where to assign each part; how to know where the various pieces of an access request should be sent; and how to reintegrate the responses to the requests. Some of these problems (e.g., the first one) have been diligently studied, but for most neither the problems nor the solutions are yet well understood.

In many DDP architectures, it is necessary to replicate decentralized data (such as target parameters). While it is possible to have centralized control over multiple copies of a data base, it is frequently unsuitable and/or undesirable owing to system integrity, reliability, performance, or

other issues. Therefore, it is critical that we learn how to implement decentralized control over a decentralized, replicated data base. The central problem is keeping the data base consistent. Some attempts to keep consistency introduce deadlock or fault recovery problems.

Problems may arise if synchronization variables are not modified in the proper sequence, because two different processors may simultaneously think that each has locked the shared variable and, thus, proceed to modify it. Updates to shared variables can also be handled by using time stamps attached to the requests or by using voting mechanisms. In fact, six different schemes have been proposed to solve this problem, but their performances are unknown.

3.2.1.1 Why Experiment?--Experiments are needed to assist analysts in approaching this problem. They need additional intuition and additional design, analysis, and proof techniques before they can effectively work on this problem. The knowledge and intuition concerning this problem are so limited that researchers do not know how to approach the proof questions--to show either that a scheme is logically correct or that it is efficient. The most useful result is one stating that a certain scheme produces the same final state in the data base after all transactions settle out.[3] As no statements can be made concerning the delays before this ideal condition pertains, the result, while theoretically useful in that it shows correctness, is not practicable. This gap is particularly important in real-time command and control applications.

3.2.1.2 Expected Results--An experiment with replicated data base systems would evaluate performance parameters of schemes proposed to solve the problem and may even uncover situations in which one or more of the schemes does not give correct behavior. This information is most essential in coordinating the use of shared information and distributing control over the data base to a number of independent processors (which is necessary for redundancy and failure-mode system operation).

3.2.1.3 Relationship with BMD Payoffs--Correct, timely management of distributed replicated data bases is essential to support BMD payoffs, including:

1. Reconfigurability
2. Programmability
3. Function modularity
4. Correctness
5. Timeliness

Reconfigurability is supported by the fact that the different copies of the data base can be used by many modules within the system. Programmability and function modularity are supported by the mechanism being decoupled from the concerns of those persons implementing functional modules within the system. Correctness of the distributed data base is essential for system correctness, and system timeliness is directly related to the speed of the coordination mechanism.

The degree of influence of these effects on overall system suitability depends, naturally, upon the usage frequency of items within the replicated data base. System designers can increase the system reliability by replicating data, but will be loath to use this technique if the attendant overhead is too great. As discussed in Volume IV, Subsection 7.3.5, the distributed designs created during our architectural development studies did not emphasize memory issues, so the replicated data base problem was not introduced into any of those designs.

3.2.1.4 Experimental Approach--A combination of simulation and emulation seems appropriate for experiments on replicated data base management. The demand characteristics could be simulated, but the detailed coordination mechanisms must be emulated in order that they show all transactions required for proper coordination.

The proposed experiments will develop confidence in new solution techniques and may develop some relative performance measures. We will compare several update synchronization solutions with respect to correctness and performance. Three possible solutions are described in the following subsections.

3.2.1.4.1 Common Update Arrival Sequences--All data base copies will agree if they are initialized identically and receive updates in the same sequence. Common sequencing can be ensured by passing all update requests to all data base modules along the same structure of serial links or by passing all copies of the data base past the modules making updates in the same sequence. The former scheme appears more attractive for several reasons, including:

1. Lower communications requirements, and

2. Better failure resiliency.

Note that while the circulating data base must pass through a loop topology of updating modules, the circulating update requests may pass through an arbitrary interconnection structure as long as the requests are not reordered.

3.2.1.4.2 Sequence Numbers--In the schemes of Subsection 3.2.1.4.1, the update requests must arrive at the data base in the updating order. The data base module could order the requests after their arrival if each request included sequence information. The big problem with this scheme is to develop efficient techniques to generate unique sequence numbers. If this could be solved in a distributed manner, the system is likely to be more fault tolerant than those discussed in Subsection 3.2.1.4.1.

3.2.1.4.3 Data Base Manager Votes--In this scheme, update requests are not ordered by arrival at a common point or by departure from a common point. Rather, they are implicitly ordered by arrival times at the data base management modules. Whenever any module attempts to perform an update, it transmits information to all other data base managers concerning the proposed update. Each data base manager votes on each update proposal by determining whether it has a conflicting update request.

Votes allow the manager to proceed or the manager must withhold the request until a later time.

3.2.1.4.4 The Experiment--These three proposed solutions to the replicated data base update synchronization problem differ considerably with respect to a number of important attributes, such as computation and protocol overhead, throughput, memory consumption, maintenance of request time sequencing, operations supported, implications on the architecture and implementation of the hardware, etc. This problem and these solutions are very poorly understood, and formal analysis shows little promise of helping. While simulations may be an effective means of evaluating these and other solutions to the problem, it is likely that coding the actual algorithms and running them with a small data base on a group of interconnected minicomputers would cost less and provide more results.

The proposed experiment will require a set of processors that simulate update request generation following statistics determined from BMD applications analysis. Processors will execute the data base updates when the policy permits.

The experiment performance will be monitored for correctness by occasionally halting request processing and examining the data bases for consistency. Performance will be determined by measuring rates of updates and the amount of communication and computational loads devoted to update synchronization.

The experimental facilities described in Section 6 will be adequate to support this experiment without requiring special tools.

3.2.2 Decentralized Resource Allocation

The problem is to efficiently and effectively allocate resources from a distributed pool of resources without requiring a central control mechanism. One proposed mechanism to solve the problem has been implemented,[4] but it is not understood and it has not been adequately instrumented.

This problem is of critical importance for DDP systems applied to BMD problems since it affects reliability, efficiency, and other important BMD payoffs, as discussed below.

3.2.2.1 Why Experiment?--Two mechanisms have been proposed: the DCS bidding scheme of Farber[5] and the virtual ring scheme of LeLann.[6] The DCS scheme has been implemented but not instrumented; the virtual ring scheme has been proposed but not implemented. Neither scheme has been analyzed, as the environment of the application is extremely complex. An appropriate, well-designed experiment would yield values of performance measures for the proposed mechanisms and uncover areas for improvement in the mechanisms. If the experiment is modularly designed, it should be possible to use the same experimental facility to test other protocols without starting from scratch.

3.2.2.2 Expected Results--The major results from this experiment will be measurements of several parameters describing the performance of particular decentralized resource allocation mechanisms. Interesting performance parameters include:

1. Processing delays experienced by resource requests.

2. Probabilities that requests will be satisfied upon the first request.

3. Probabilities that requests cause deadlock situations which preclude their satisfaction.

4. The processing required in each module to handle each request.

5. The intermodule communications requirements for each request.

The mechanism performance is strongly dependent upon the characteristics of the environment, particularly the statistical patterns of allocation requests. Therefore, an experimental outcome would be a family of performance curves with the parameters of the allocation request statistics as independent variables.

3.2.2.3 Relationships with BMD Payoffs--The efficiency of resource allocation mechanisms affects the performance of a DDP system in a BMD environment through several payoffs, including capacity, timeliness, adaptability, and survivability. The system capacity is adversely affected by increased delays in the resource allocation mechanism. Similarly, timeliness is adversely affected by increased resource allocation overhead. In many applications these effects can be small, because most resources can be allocated when the system is designed. One must realize, however, that the overhead must be incurred whenever any amount of any resource

(that is managed in a distributed manner) is required. Adaptability and survivability are intimately related to distributed resource management because any changes in system structure must be handled by a distributed policy (as the combinational growth of the number of possible system configuration changes requiring resource management precludes managing the resources by some policy fixed before the system is in operation).

3.2.2.4 Experimental Approach--These experiments should be performed by emulation, since we want to understand the performance of the management algorithms, and we want to look for any pitfalls hidden in the algorithms (which have not been studied extensively). The emulations of the application environment should be clearly separated from the emulations of the management algorithm. This permits the same emulation structure to be used to evaluate other management algorithms that may be proposed. Statistically-generated inputs will drive the emulator. Results are obtained by counting the times required for the requests to propagate through the system and be satisfied.

The experimental facility described in Section 6 will be adequate to support this experiment.

3.2.3 Decentralized Scheduling

Decentralized scheduling is similar to decentralized resource allocation in that a system control function is being performed in a decentralized manner. Unlike decentralized resource allocation, there are no proposed schemes for implementing general decentralized scheduling using any policies. A scheme proposed by Stone[7] has been shown to apply only to

the cases of two or three processors; it has also been shown that this scheme cannot be generalized in any way to systems with four or more processors.

In the absence of proposed algorithms or scheduling schemes, we can propose scheduling for real-time systems where the set of tasks to be performed is fixed beforehand. In this case, one may establish a set of schedule templates (with some potential for limited modification) using some generating algorithm. Experiments to ascertain the performance of the template schedules generated by a particular algorithm would increase our understanding of how decentralized scheduling might be achieved in a more flexible manner at some future time.

3.2.3.1 Why Experiment?--We need to experiment with scheduling mechanisms to better understand the problems attendant upon decentralized scheduling. Experimental experience will help us develop our intuition and provide a basis for future algorithm proposals. Fixed template scheduling must be studied first, as we do not have any proposed mechanisms for flexible distributed scheduling that we might use as a basis for experimentation.

3.2.3.2 Expected Results--Scheduling experiments can quantify the overhead in the scheduling operation and can be used to determine how well the scheduling policy meets given system performance criteria. Minimum delays for high priority tasks may be a good criterion. (Note that one cannot use the average delay, averaged over all tasks in the system, as this number is affected only by the scheduling overhead and not by the order in which tasks are in fact scheduled.)

3.2.3.3 Relationship with BMD Payoffs --The BMD system must respond quickly to unanticipated external events. It is difficult to meet this requirement by using any fixed scheduling policy. Thus, we require a flexible scheduling mechanism that would contribute to response timeliness and system capacity. A flexible mechanism is essential to cope with node, subsystem, and module failures.

3.2.3.4 Experimental Approach--We must emulate proposed scheduling mechanisms to study their performance within specific demand environments. A modular design would incorporate scheduling emulation modules coupled to environment simulation modules. The experiment would measure the scheduling delays and instrument the details concerning where the requests spend their time within the system (i.e., where there may be bottlenecks). As we do not have any proposed solutions to this problem, we cannot actually perform this experiment; we can, however, design an experimental facility into which we could easily couple emulations of future designs.

The facility described in Section 6 will adequately support these experiments.

3.2.4 Loop Protocols

Protocols control information flow along communications links interconnecting the elements comprising a distributed system. Numerous protocols have been proposed to solve this control problem for various interconnection techniques; they differ with respect to overhead and transmission delays. An efficient control mechanism is required in BMD and other fast-response real-time systems to ensure timely responses to external events.

In this subsection, we describe a loop protocol experiment; a similar experiment concerning bus protocols is discussed in Subsection 3.2.5.

3.2.4.1 Why Experiment?--No loop protocol has been analytically modeled; one has been simulated[8] but it did not use any real-time environment considerations. Experiments can clarify the relative performances of the schemes under controlled environments, producing design criteria that can be used while configuring DDP systems.

3.2.4.2 Expected Results--This experiment will produce data points on design trade-off curves. For each protocol the trades between overhead, bandwidth, and error detection capability should be quantified.

3.2.4.3 Relationships with BMD Payoffs--Reliable, high-bandwidth communications assist modularity, capacity, responsiveness, and survivability. Allocation flexibility and timeliness can be problems in loop interconnections. In real-time applications, the sequence in which the computers are allowed to place messages on the loop should not be constrained by the physical order in which they are connected in the loop. Furthermore, it should be possible to assign different proportions of the loop bandwidth to various computers in order to match any application throughput nonuniformity. These two requirements should be met with minimum delays in loop allocation or message transit around the loop. Previously, it had not been possible to allocate loop usage in a way which is both flexible and timely for real-time applications.

3.2.4.4 Experimental Approach--The experiment we propose will use functional simulation on a network of small computers to determine the performance parameters of two loop protocol schemes. The results will be useful in detailed designs.

Each processor connected in a loop is interfaced through an interface module (see Figure 2). Typical loop interface functions include deciding when a message from its associated computer can be placed on the loop, deciding when a message arriving from its neighboring interface is intended for its associated computer, acknowledging correct or incorrect receipt of a message intended for its associated computer, and removing from the loop a message which has been correctly received at its ultimate destination.

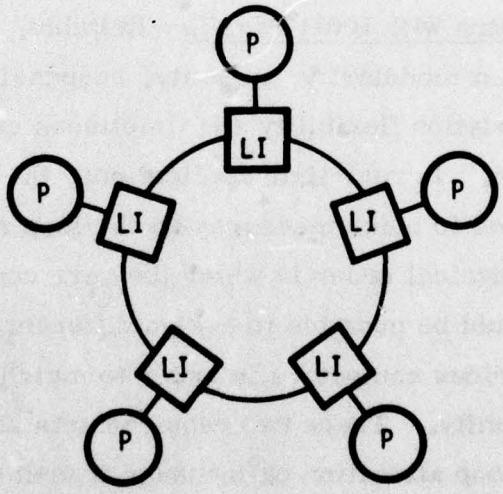


Figure 2. Typical Loop Architecture

We describe two proposals for efficient, timely loop protocols before we return to the experiment itself.

3.2.4.4.1 The DLCN Scheme--In the DLCN system[9] each loop interface unit contains a buffer for information circulating around the loop. Any message shorter than the empty buffer space can be transmitted by filling the buffer rather than relaying bits coming around the loop. When an unused space arrives, the buffer can be emptied. This scheme offers advantages over the Pierce loop[10] and the Newhall loop[11] in reducing queueing delays.[12] It is not yet clear whether the DLCN approach is suitable for real-time systems, but it is one of the prime candidates.

3.2.4.4.2 The Honeywell VDPA Scheme--A different approach to overcoming loop deficiencies has been invented by Honeywell. It achieves the allocation flexibility described above with the minimum allocation and transit delays theoretically possible. The method is an extension of an improved allocation mechanism (VDPA) invented by Honeywell for buses and employed in the Honeywell Experimental Distributed Processor.[13]

The following is an overview of the improved loop allocation mechanism. Each loop interface unit (LIU) contains an N-bit binary allocation vector (AV), where no two AVs have a "one" in the same position (see Figure 3). Each LIU contains an allocation vector pointer (AVP), with all AVPs initialized to zero. All AVPs are incremented by one (modulo N) after each message slot time; the LIU whose AVP then points to a "one" in its AV has the opportunity to transmit on the loop. Note that the loop transmission opportunity may be accepted or refused (in which case the loop is reallocated). The AVs are organized like the HXDP bus AVs.

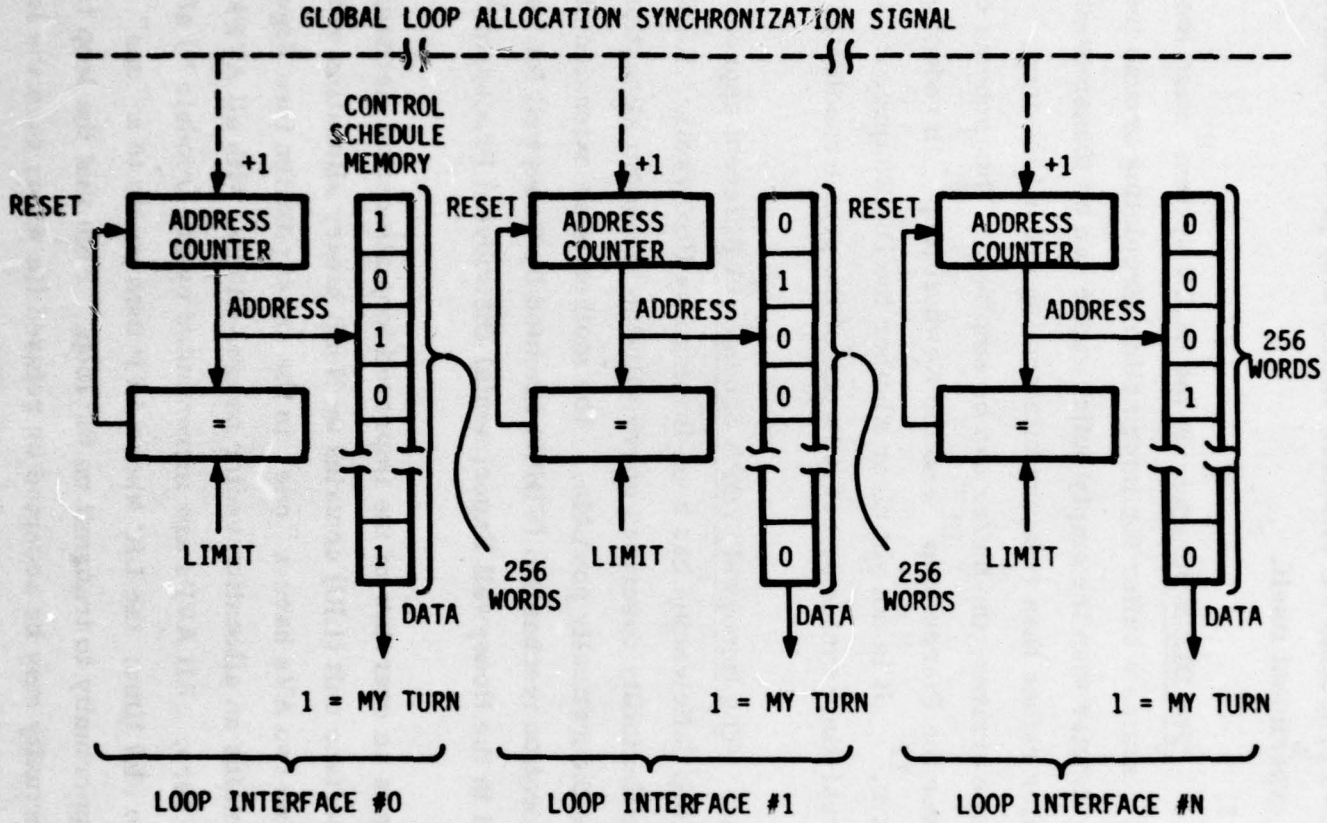


Figure 3. VDPA Loop Allocation Mechanism

This allocation mechanism could be used directly for a loop as follows. To minimize loop transit delay, the L loop interfaces should have only one bit of storage each. To transmit, the source loop interface places an M-bit message onto the loop. The ultimate destination loop interface copies (but does not remove) the message, and finally the source loop interface removes the message from the loop. It also must replace the earlier bits with the later bits modulo L and replace the last L bits with zeroes. This assumes that M>L (the usual case); otherwise, all message bits are replaced with zeroes. This naive approach does not really work well; although allocation flexibility is achieved, there is still a timeliness problem because an entire loop transit time (L bit times) is needed for all loop interfaces to agree that a message has ended. Thus, even if loop interface I is assigned two loop transmission opportunities with loop interface J inbetween, L bit times are wasted before I discovers that J refused the intervening opportunity. This wastes loop bandwidth and degrades subsystem response time. The loop allocation proposed for the experiment is an improved method.

The new method reduces I's waiting time to only D bit times, where D is the number of zeroes separating two successive ones in I's allocation vector. Now, after transmitting a message, I sends D zeroes and then begins sending its next message, it an attempt to fully utilize the loop. During the D zeroes or the first L bits of its next message, if I discovers that a loop interface assigned one of the D intervening loop transmission opportunities chose to transmit, then I stops sending its own message (to be sent later) and begins forwarding the incoming message, which came from the current "owner" of the loop. A loop refusal is a zero bit; messages

begin with a one bit. The implementation of the improved loop allocation mechanism is shown in Figures 4, 5 and 6. It is similar to the HXDP bus allocation mechanism, but uses two allocation vector pointers for receive and transmit decisions.

3.2.4.4.3 The Experiment--Briefly, the loop algorithms experiment calls for individually emulating both loop algorithms (both DLCN and VDPA) on a general loop topology test facility, applying a specified set of communication demands to each system, monitoring, and analyzing the resulting performance data.

The hypothesis for this experiment is that the HI system will perform better with respect to throughput and response time for certain system communication loads, especially loads asymmetric as to source terminals, which include timely message delivery as an important system requirement.

The test facility for this experiment would consist of a set of small processors (PDP 11/03's are used in the example) connected in a loop using standard moderns and modern interfaces (see Figure 7). In this configuration, each 11/03 would serve two functions: emulation of the particular loop algorithm being considered and functional simulation of the traffic pattern of its associated architectural element (assumed to be a processor in the preceding discussion, but possibly an I/O device). This combination of simulation and emulation is chosen because any loop interface algorithm can be implemented and tested easily, both to verify the correctness of the algorithm and to determine its performance characteristics.

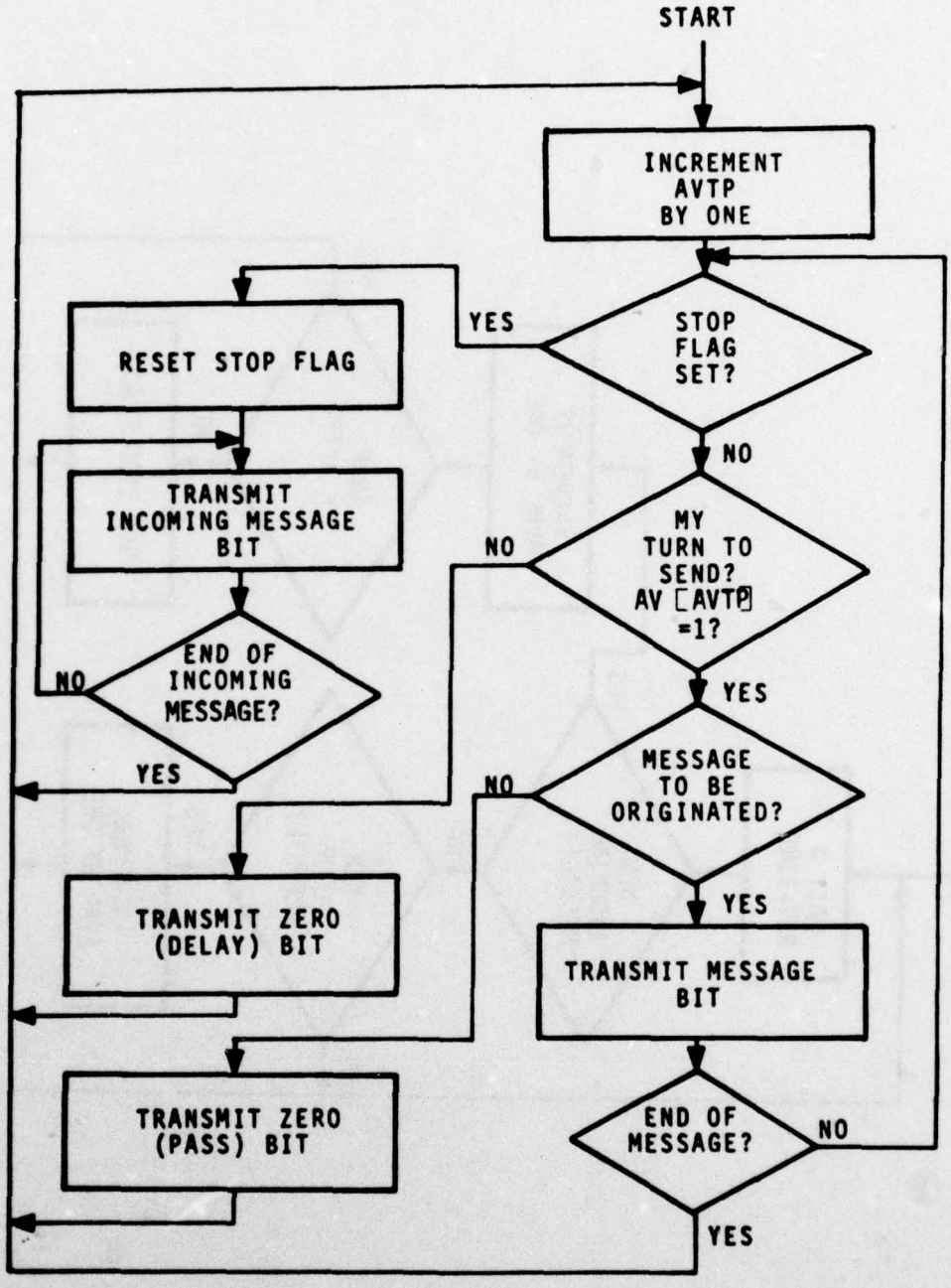


Figure 4. VDPA Loop Allocation Mechanism-- AVTP Control Flow Chart

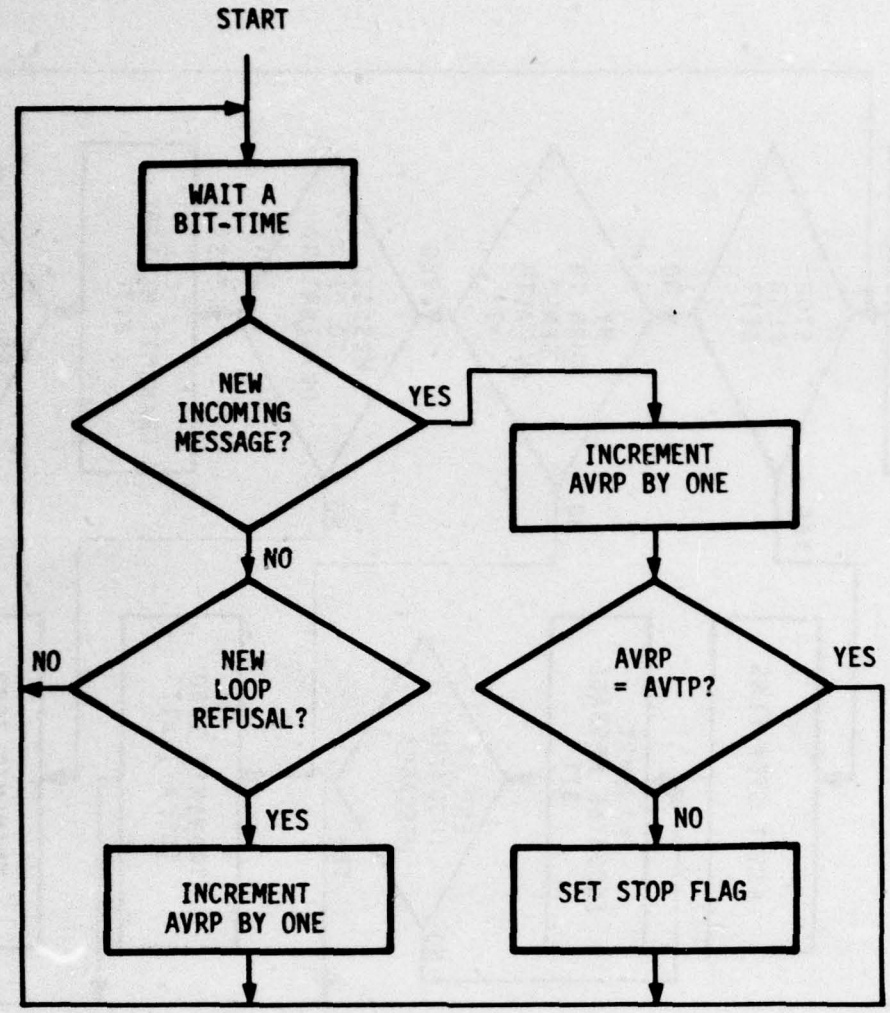


Figure 5. VDPA Loop Allocation Mechanism--AVRP Control Flow Chart

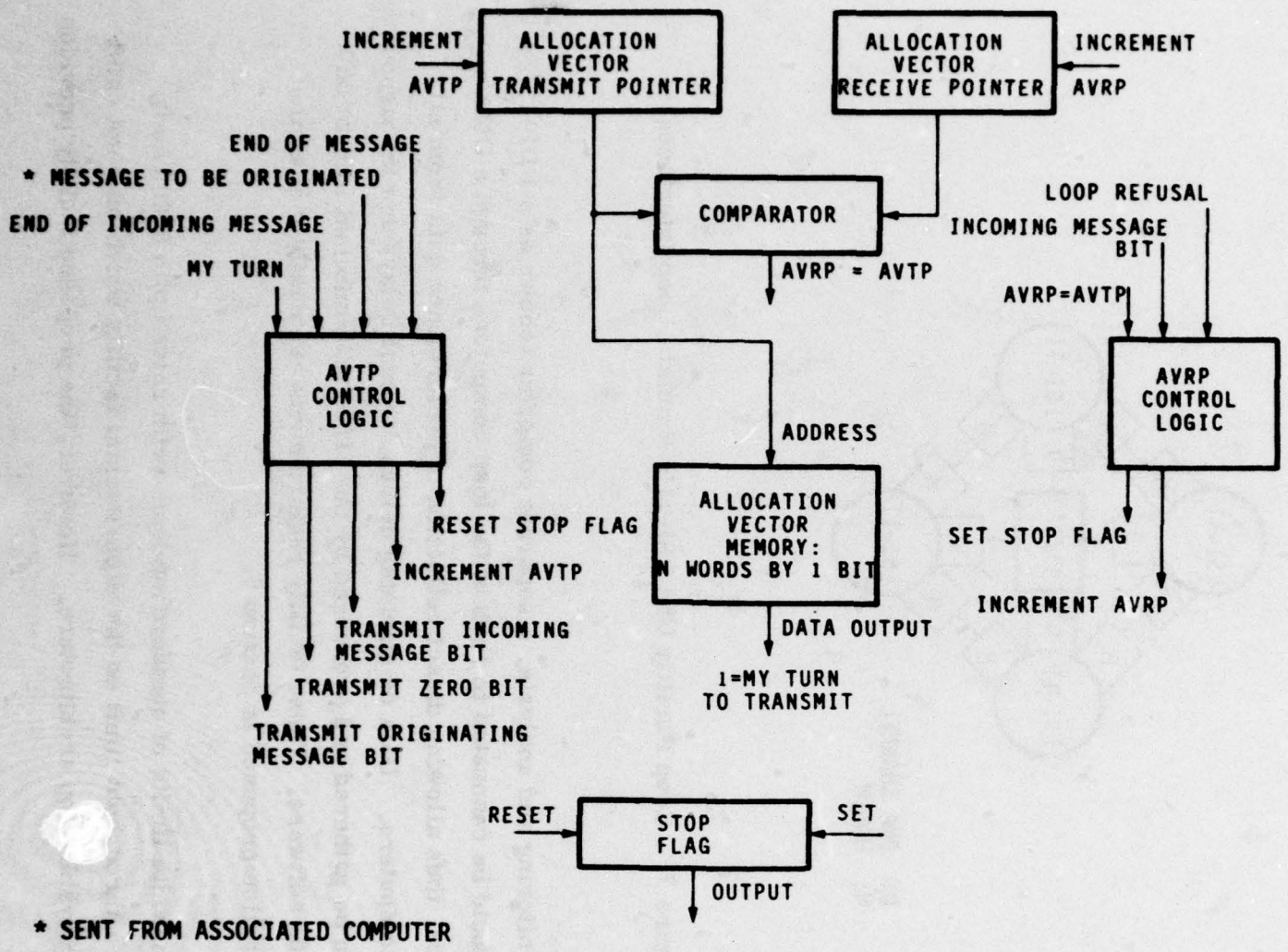


Figure 6. VDPA Loop Allocation Mechanism -- Block Diagram

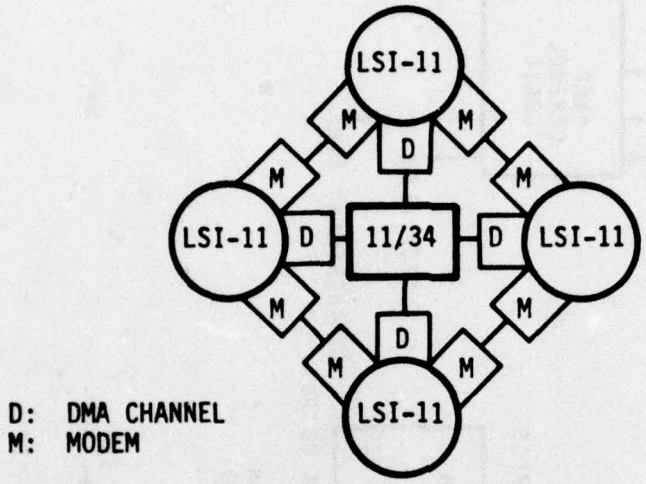


Figure 7. Loop Facility (the LSI-11's Emulate Loop Interfaces)

For monitoring and analysis, a separate computer (shown as a PDP 11/34 here) would be connected to each of the loop computers through a DMA interface, thus allowing direct collection of performance data from all small computers. Data on queueing delays, transit delays and throughput rate can be gathered and processed by the 11/34 with minimal impact on the 11/03 software. These facility requirements are easily met by the configuration proposed in Section 6.

The bandwidth limits of standard modems (with rates up to 9600 baud) impose a throughput limit on the experimental facility which may not exist in a deployable loop architecture. However, the proposed facility provides

an inexpensive, readily available loop architecture which allows the experimenters to concentrate on the control algorithms without requiring a detailed simulation of the loop topology. It also allows the use of relatively slow, cheap processors to emulate the loop interface units; faster processors or hardwired units would normally be required to keep up with "normal" loop rates, which may be above 1 Mbps.

3.2.5 Bus Protocols

Much of the preliminary discussion in Subsection 3.2.5 concerning loop protocols applies without modification to bus protocols; it is repeated here with appropriate minor changes for completeness.

Protocols control information flow along communications interconnecting the elements comprising a distributed system. Numerous protocols have been proposed to solve this control problem for bus systems. With a centralized controller, the problems are well understood, but more work remains for buses using distributed control mechanisms. Though simple models can be used to bound shared bus system performance[14], experiments are needed to determine realistic bounds. Different bus protocols produce different overhead and transmission delays. Efficient control mechanisms are needed for BMD and other fast-response real-time systems requiring timely responses to external events. Later experiments could investigate techniques to control multiple-bus interconnection schemes.

3.2.5.1 Why Experiment? --The CSMA protocol has been analyzed[15], but the VDPA protocol[16] has been implemented without modeling. Experiments can clarify the relative performances of the two schemes in similar environments, producing design criteria that can be used while configuring DDP systems. If the same environments are also used in the loop experiments (Subsection 3.2.4), meaningful comparisons between bus and loop interconnections can be developed.

3.2.5.2 Expected Results --This experiment will produce data points on design trade-off curves. For each protocol, we quantify the trades between overhead, bandwidth, and error detection capability. These trades can be compared with similar information concerning loop protocols (see Subsection 3.2.4.2).

3.2.5.3 Relationships with BMD Payoffs--Reliable, high-bandwidth communications assist modularity, capacity, responsiveness and survivability.

3.2.5.4 Experimental Approach--This experiment will determine whether the Carrier Sense Multiple Access (CSMA)[17] allocation scheme has significant advantages over the Vector Driven Proportional Access (VDPA)[18] allocation scheme. We describe these allocation mechanisms before returning to the experiment itself.

3.2.5.4.1 The Honeywell VDPA Scheme--The VDPA allocation scheme for buses is very similar to the Honeywell VDPA loop control scheme discussed in Subsection 3.2.4.4.2. Each bus interface unit (BIU) contains an allocation vector (AVP) as before (see Figure 3). At the end

of each message, all BIUs increment their AVPs, and only the unique BIU with a "one" in the word selected by its AVP may send a message. If the selected BIU has no message to send, it instead puts a short reallocation signal on the bus, causing all BIUs to again increment their AVPs.

The advantages of the VDPA scheme include the following: bus capacity can easily be apportioned to BIUs at system configuration time, and reallocation (when the selected BIU has no message) is quick and does not incur the overhead of a central control device. It is also possible to modify the AVs on line, but this strategy can introduce many problems if the system is operating in an error-prone environment.

3.2.5.4.2 The CSMA Scheme--The CSMA scheme we propose here, called prioritized CSMA, is an adaptation of previous work on bus allocation in a non-real-time environment. The CSMA method does not attempt to synchronize bus allocation as do other schemes. Instead, when a BIU has a message to send, it simply sends it immediately, unless the bus is already busy, in which case it waits until the bus is idle. Two messages may "collide" because they were transmitted onto the bus by two BIUs that sensed an idle bus. The collision is detected by failures of redundancy checks included in the messages. This failure causes the acknowledge signals not to be returned to the sending BIUs. This absence is detected by a fixed time-out in each BIU.

When a collision occurs, each BIU that did not receive an acknowledgement generates a different pseudo-random time-out value from a probability distribution with controlled parameters, which may depend upon the priority of the message and the recent collision history experienced by the

BIU. The BIU then counts this value down to zero and attempts to retransmit the message after the time-out reaches zero. If there are repeated collisions, each involved BIU modifies the parameters of its time-out counter distribution function to cause the count values in the competing BIUs to diverge in a controlled way. The algorithm each BIU uses to change its distribution depends upon the priority of the BIU, the priority of the message, its past traffic history, and other factors.

The CSMA method offers considerable allocation flexibility and timeliness. It is flexible because the system will automatically adapt to new traffic demands without any explicit reconfiguration commands. It offers timeliness because allocation is primarily demand driven, allowing a terminal with an outbound message to transmit it immediately.

3.2.5.4.3 The Experiment--Emulation experiments are required because simulation (to a sufficient level of detail) is too expensive, and because we cannot formally analyze the HXDP scheme.

In the experiment, we would individually emulate both bus control schemes on a general bus topology test facility. A specified set of bus traffic loading would then be applied to each system, first, to verify the correctness of the algorithm, and second, to monitor the system performance.

The hypothesis for this experiment is that the prioritized CSMA system will perform as well as HXDP with respect to throughput and system response time with more allocation flexibility than HXDP. The payoff to BMD will be the identification of the system with better modularity, timeliness, and capacity.

The experiment procedure will involve coding, testing, and verifying correctness of the two bus control algorithms discussed. Each will then be exercised with the same set of bus traffic loading patterns. Various loads, designed both to uncover parametric sensitivity and to represent some projected deployed system loads, will be run and statistics will be gathered.

The test facility for this experiment would consist of a set of small computers connected via a common bus as shown in Figure 8 with a common memory and more capable processor also on the bus for monitoring and possible bus control. The small computers serve two purposes: emulation of BIUs by executing BIU algorithms and generation of bus traffic loads. This facility approximates, so much as is possible with off-the-shelf hardware, the capabilities of a shared bus. The main processor

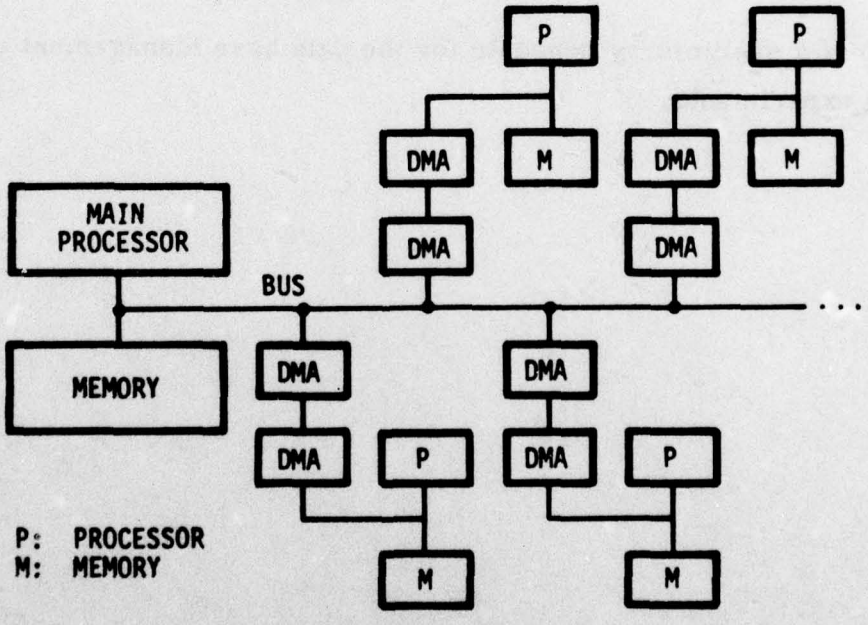


Figure 8. Bus Experiment Facility

can be used to easily simulate specific bus system attributes such as transport delay values and bus allocation characteristics, as well as do monitoring, data processing, and error injection.

The facilities described in Section 6 will be adequate to perform this experiment.

3.3 EXPERIMENT PRIORITIES AND SCHEDULE

We place highest priority on the data base management experiment (Subsection 3.2.1) and next priority on the loop and bus protocol experiments (Subsection 3.2.4 and 3.2.5). The resource allocation and scheduling experiments require further research and development during the next phase of the project and thus are not candidates for early execution.

Figure 9 is a preliminary schedule for the data base management and loop protocol experiments.

DATA BASE MANAGEMENT

- DEVELOP SIMULATOR
- RUN SIMULATIONS
- CONFIGURE EMULATION SYSTEM
- DEVELOP EMULATOR
- RUN EMULATIONS
- EVALUATE RESULTS

LOOP PROTOCOLS

- CONFIGURE EMULATION SYSTEM
- DEVELOP EMULATOR
- RUN EMULATIONS
- EVALUATE RESULTS

BUS PROTOCOLS

- CONFIGURE EMULATION SYSTEM
- DEVELOP EMULATOR
- RUN EMULATIONS
- EVALUATE RESULTS

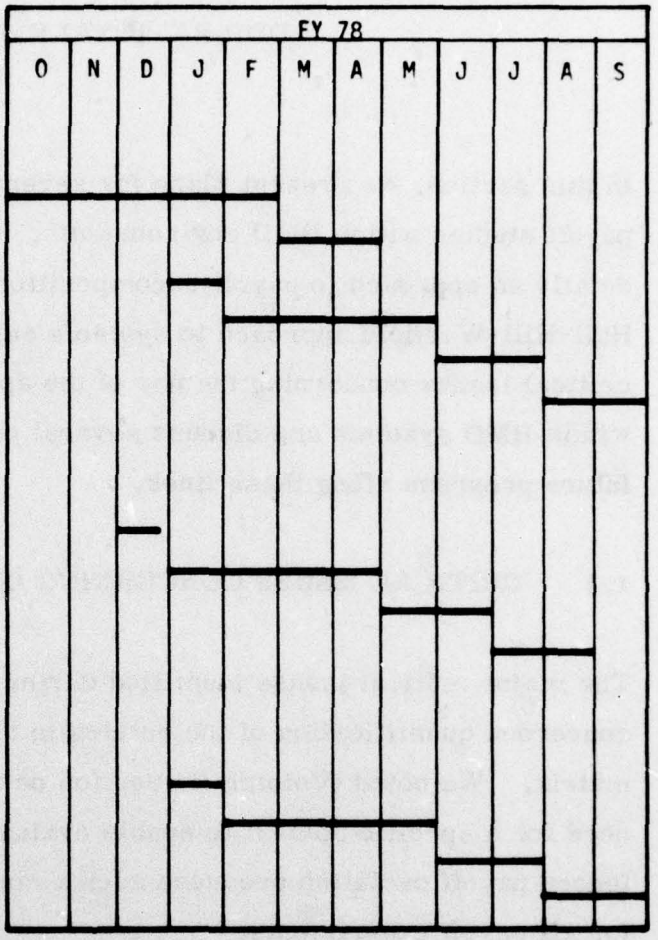


Figure 9. Proposed Schedule for Mechanism Experiments

SECTION 4

DDP RATIONALE STUDIES

In this section, we present plans for several experiments to refine DDP payoff studies within BMD environments. Volume II of this document details an approach to payoff decomposition and evaluation built on the Hall-Hill-Warfield approach to systems engineering. Here we review the critical issues concerning the use of the approach to design DDP subsystems within BMD systems and discuss several payoff experiments to assist future progress along these lines.

4.1 CRITICAL ISSUES CONCERNING DDP RATIONALES

The major critical issues identified during the study of DDP rationales concerned quantification of the entries in the means-ends cross-interaction matrix. We noted (Volume II, section on Further Research) the pressing need for a specific context to enable evaluation of the interactions. The forced payoff evolution produces such a context and is thus a prerequisite for all payoff experiments.

4.2 EXPERIMENTS CONCERNING DDP RATIONALES

For these experiment descriptions, we also assume that the preliminary system design and forced payoff evolution will identify needs for processors in different environments, such as satellites, missiles, and ground installations. Each platform forces a different set of constraints on the

DDP subsystem designer and therefore results in different payoff characteristics.

Some cross-interaction matrix elements can be determined by experimentation, but many others cannot, either because the experiment would be too costly (involving, for example, a complete DDP system design to obtain one data point) or because we do not know how to quantify the desired attribute. We restrict the following set of experiments to those that do not have glaring deficiencies in these areas.

We selected cross-interaction matrix entries for experimentation by first choosing payoffs that could be quantified and then by selecting means provided by DDP. We then chose an entry where we expect a strong positive interaction between the means and the payoff. The chosen quantifiable payoffs were:

- fault detection ease
- reliability
- fault isolation ease

4.2.1 Fault Detection Ease with Small Processors

We wish to determine the relationship between processor size and the ease or difficulty of detecting faults in the processor. Two approaches suggest themselves; their relative usefulness for BMD problems can be determined based on whether one wishes to ask about special-purpose processors or not.

If one wants to know only about general-purpose processors, then one can study existing processor designs to determine the difficulty in fault detection; no experimentation is needed. If, on the other hand, one wants to obtain this data concerning special-purpose designs, then one must experiment by designing several such processors of varying sizes. This method is best accomplished by picking specific BMD functions and assigning designers to develop architectures for those functions. It might be particularly interesting if the designers are asked to emphasize fault detection during their design exercise.

Once we have a processor design, we can estimate the trade between additional hardware for fault detection and the length of test sequences to test for faults in the processors. Test length should provide an adequate measure of detection difficulty, assuming that all faults can indeed be detected within the test. If there exist faults that cannot be detected by the test, a penalty must be applied to the test length measure. The nature of this penalty has not been studied.

4.2.2 Extra Gates Improve Reliability

The relationship between adding extra gates and improving the reliability is similar to the relationship between decreasing processor size and improving the fault detection ease: both relationships can be verified either by paper studies or by experiments designed to produce data points on trade-off curves. Any experiment to measure reliability must incorporate some external means of stressing the system so that the MTBF is reduced sufficiently that the experiment may produce statistically significant results in reasonable time.

In this particular experiment, designers would be asked to add gates to existing system designs, and the reliability of the composite system would be compared with the reliability of the original system. This experiment would increase our understanding of the relation between reliability and redundancy and also develop new techniques for adding redundancy to improve reliability.

4.3 SUMMARY

Rationale studies require much further research and development before we can expect to refine the notions sufficiently that meaningful experiments can be specified. The preliminary sketches of experiments presented in this section should serve as a basis for refinements, but it would be premature to assign priorities and schedules to these proposals. In any event, no meaningful rationale experiment seems possible before FY 79.

SECTION 5

THROUGHPUT SIMULATION OF A DISTRIBUTED DATA PROCESSING SYSTEM

In this section we describe the simulation of a distributed data processing system by means of Honeywell's Generalized Computer System Simulator to determine the system's throughput efficiency. The Modular Missile-Borne Computer (MMBC) designed at Honeywell Systems and Research Center was used as an example of a logically-distributed data processing system operating in a real-time command and control environment. The experiment was performed under this contract; the example system was selected as a handy, well-understood example of a DDP system in a BMD-type environment.

The general problem is to compare alternative computer architectures and evaluate them in an objective, quantitative manner. If one were to evaluate computer architectures, the criteria would include throughput efficiency, reliability, and life cycle cost. The simulator should be useful in the evaluation of throughput efficiency.

The problem which has been addressed thus far is the evaluation of a distributed data processing system. A single architecture has been modelled. Two parameters of importance to the designer have been varied in order to evaluate and refine the design.

5.1 THE SIMULATOR

An overview of the simulator is presented in Figure 10. The hardware is described in terms of:

- Processing elements
- I/O devices
- Memory devices
- Buses

The software is described in terms of program modules with an execution time or a number of instructions representative of the system which is simulated. The connection of the software modules may be a chain of successors, or it may include an "OR" branch with assigned probabilities, or it may include an "AND" branch. The output will provide the basis for comparative evaluations of throughput, bus contention, I/O conflicts, and storage utilization. The capability of the simulator is summarized in Figure 11.

The generalized computer system simulator is written in SIMSCRIPT with FORTRAN IV I/O routines for execution on the Honeywell 6080 digital computer. A detailed description of the current version of the simulator is presented in a contract report by Arnold and Schlicht.[19]

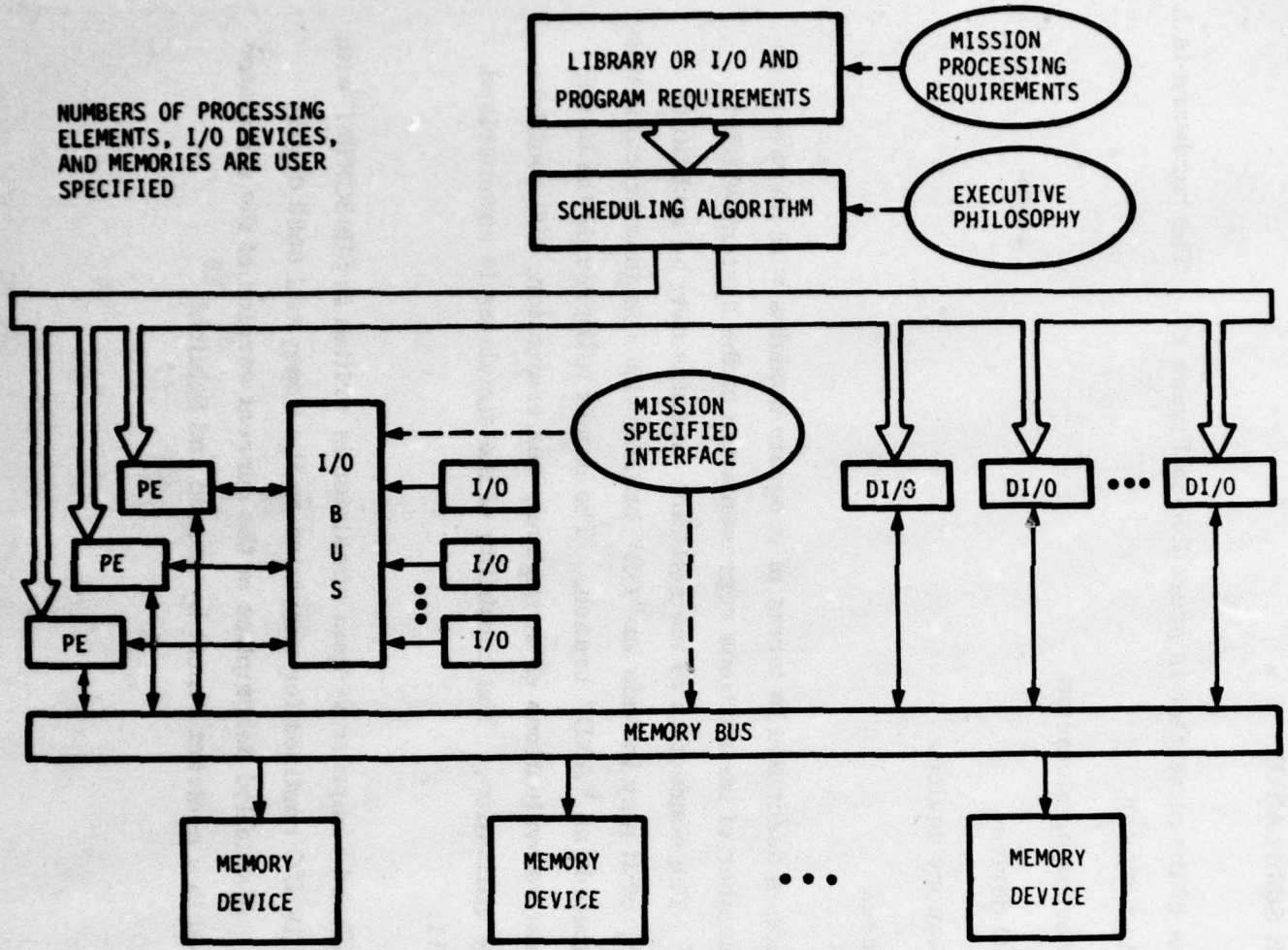


Figure 10. Generalized Computer Systems Simulator

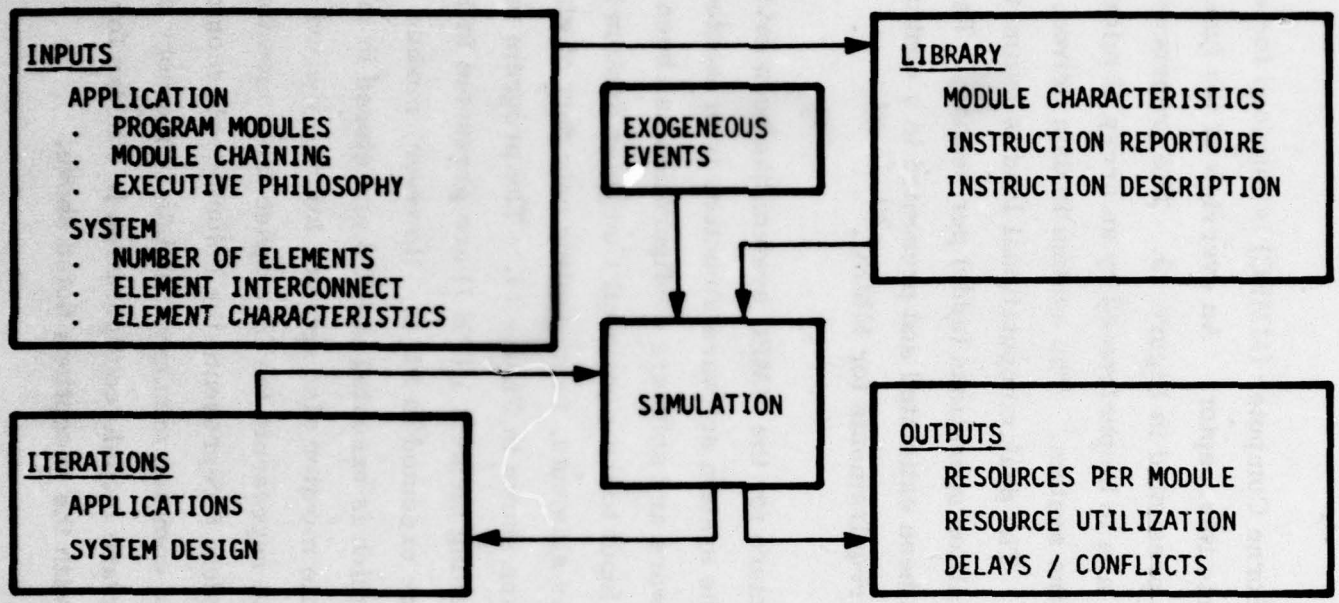


Figure 11. Computer Simulation Capabilities

5.2 THE DISTRIBUTED DATA PROCESSING SYSTEM

The Modular Missile-Borne Computer (MMBC) is intended for use in an advanced exoatmospheric interceptor. An overview of the functional flow for the mission is presented in Figure 12. The preprocessing and bulk filtering functions are to be performed by an array of microprocessors in the generic or baseline system. The system is data driven, and the data rate is very high. The peak computational load is estimated to be on the order of 31 to 43 million operations (adds) per second. The computational load has been estimated and presented in a contract report of the data processing requirements for MMBC.[20]

A preliminary architecture for the MMBC system has been devised by Dr. R.G. Arnold. The system software structure is presented in Figure 13. The hardware and software configuration has been described in a manner suitable for input to the Honeywell Computer System Simulator (HCSS). Six processor elements, representing only part of what is required for MMBC, are shown in Figure 14. The program modules which are allocated to processing element 1 (PE 1) are presented in Figure 15. Most of the modules are executed in PE 1. However, module 14 represents the data file monitor which is executed in PE 3 and stored in memory device 3 (MD 3). The data file monitor also appears later in the job stream as module 17. Module 13 represents the free space table monitor which resides in MD 2. Module 6 represents the column table monitor which resides in MD 7. The common memory consists of memory devices 7 through 17. The software module configuration is similar for all six processor elements, with the exceptions noted above.

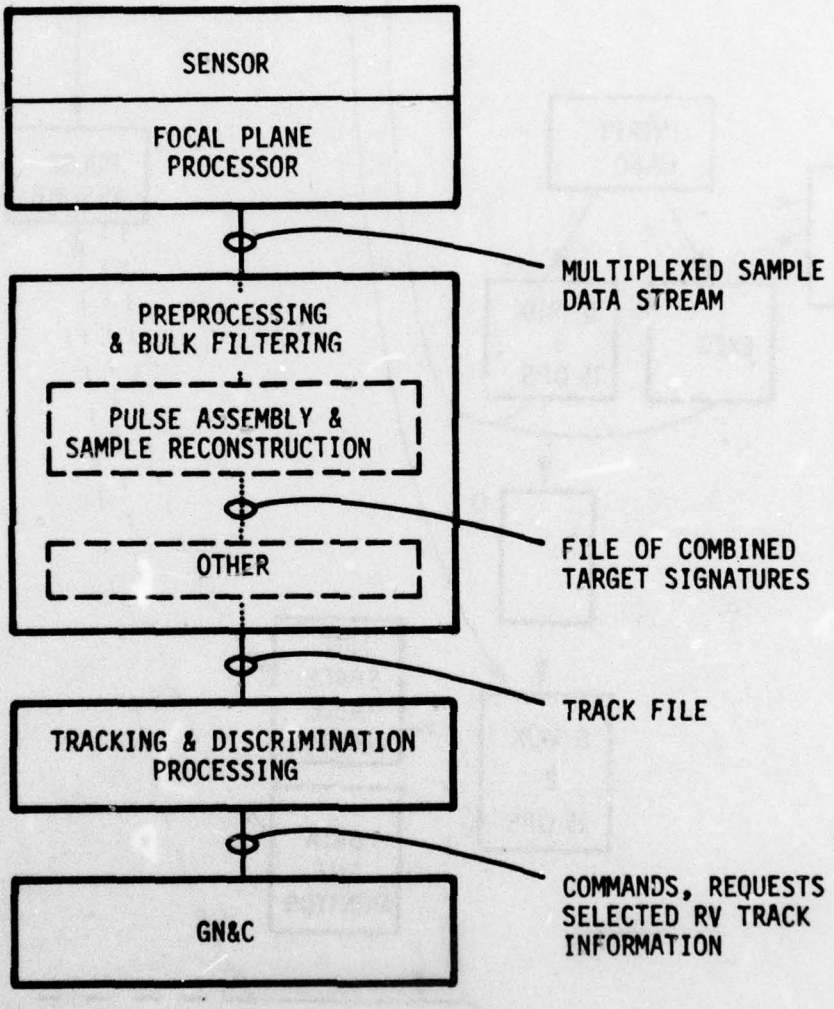


Figure 12. First Subdivision of Preprocessing and Bulk Filtering Processor Functions

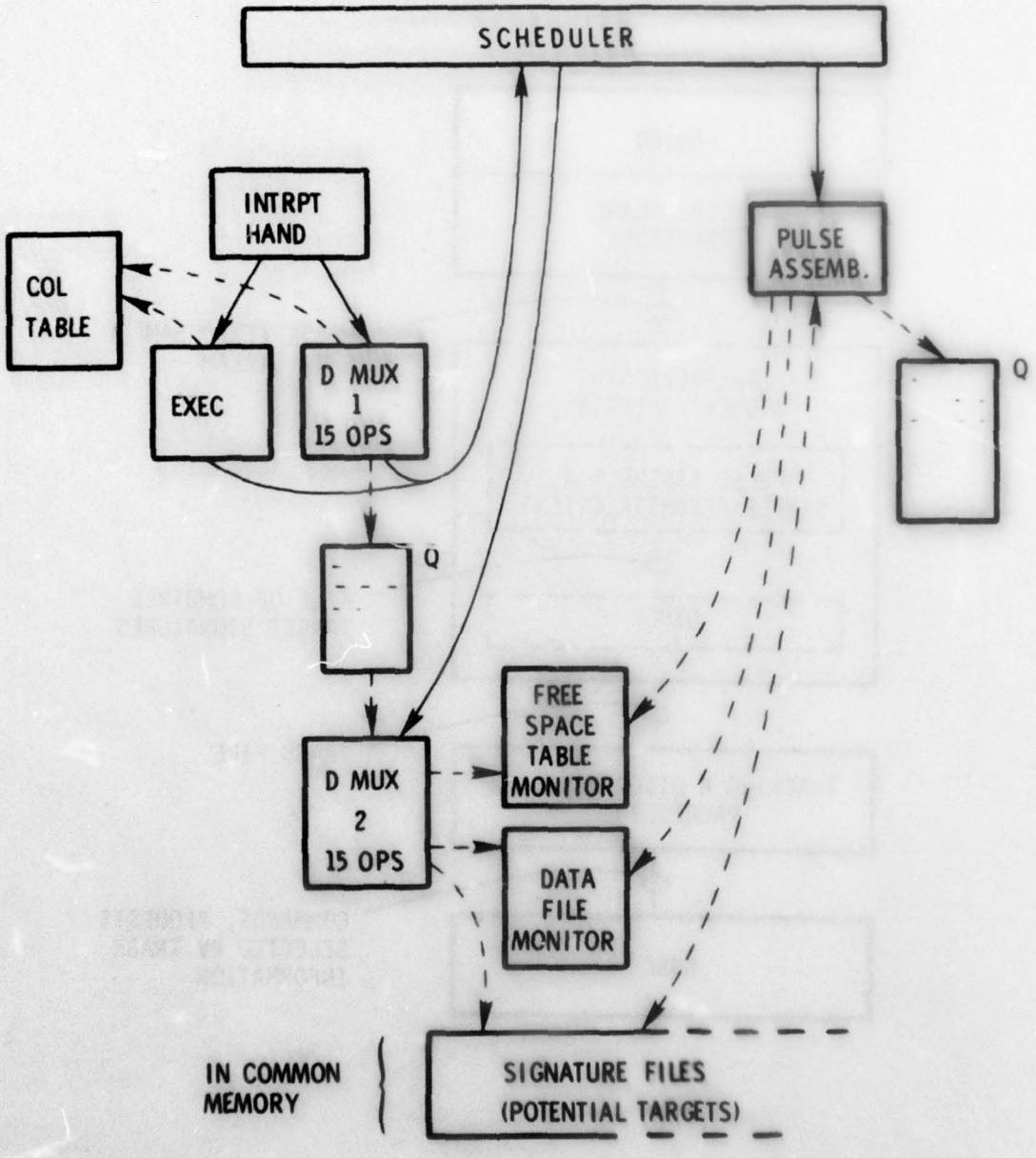


Figure 13. Modular Software Structure--Processors 1 through 6

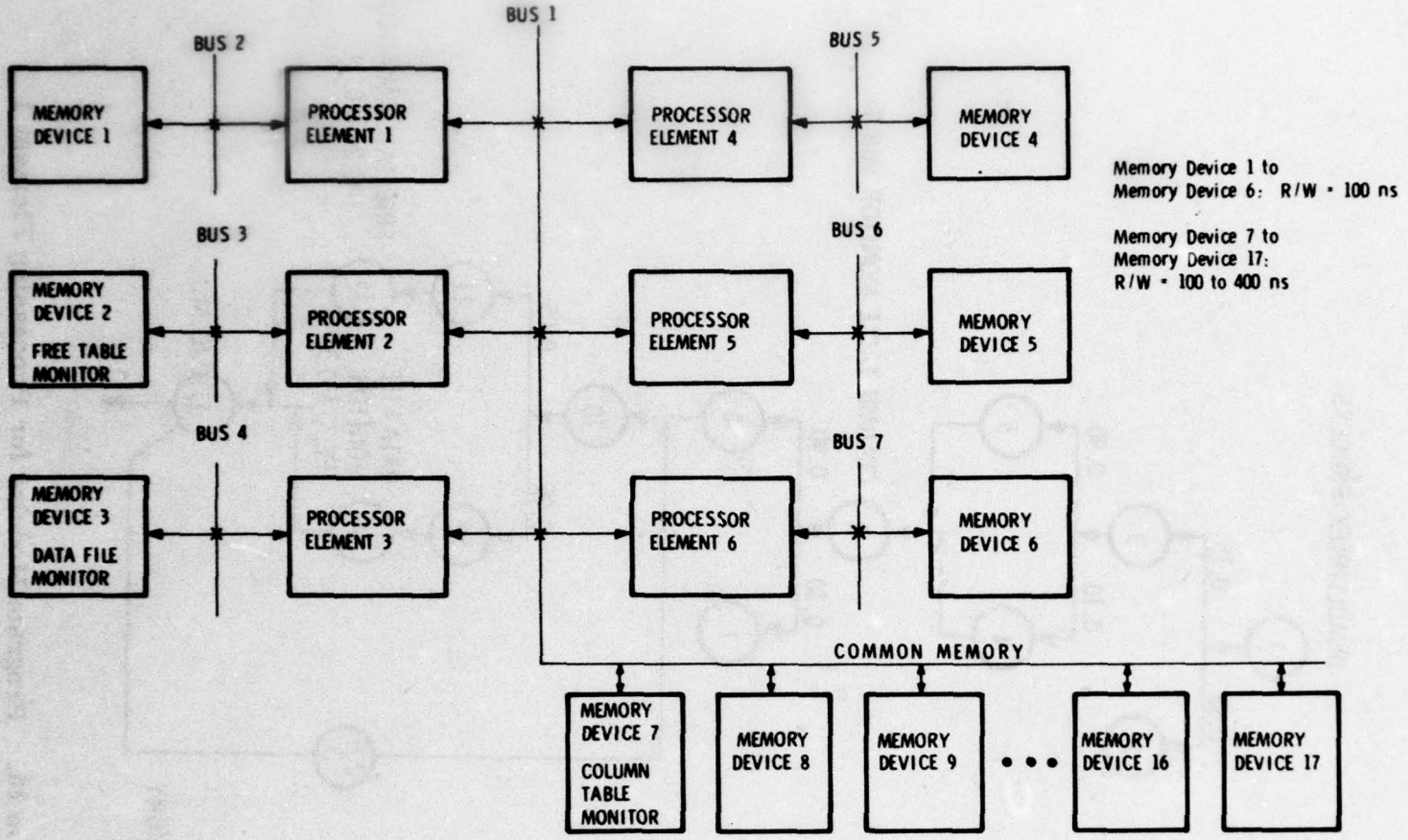


Figure 14. Simulated Hardware Configuration

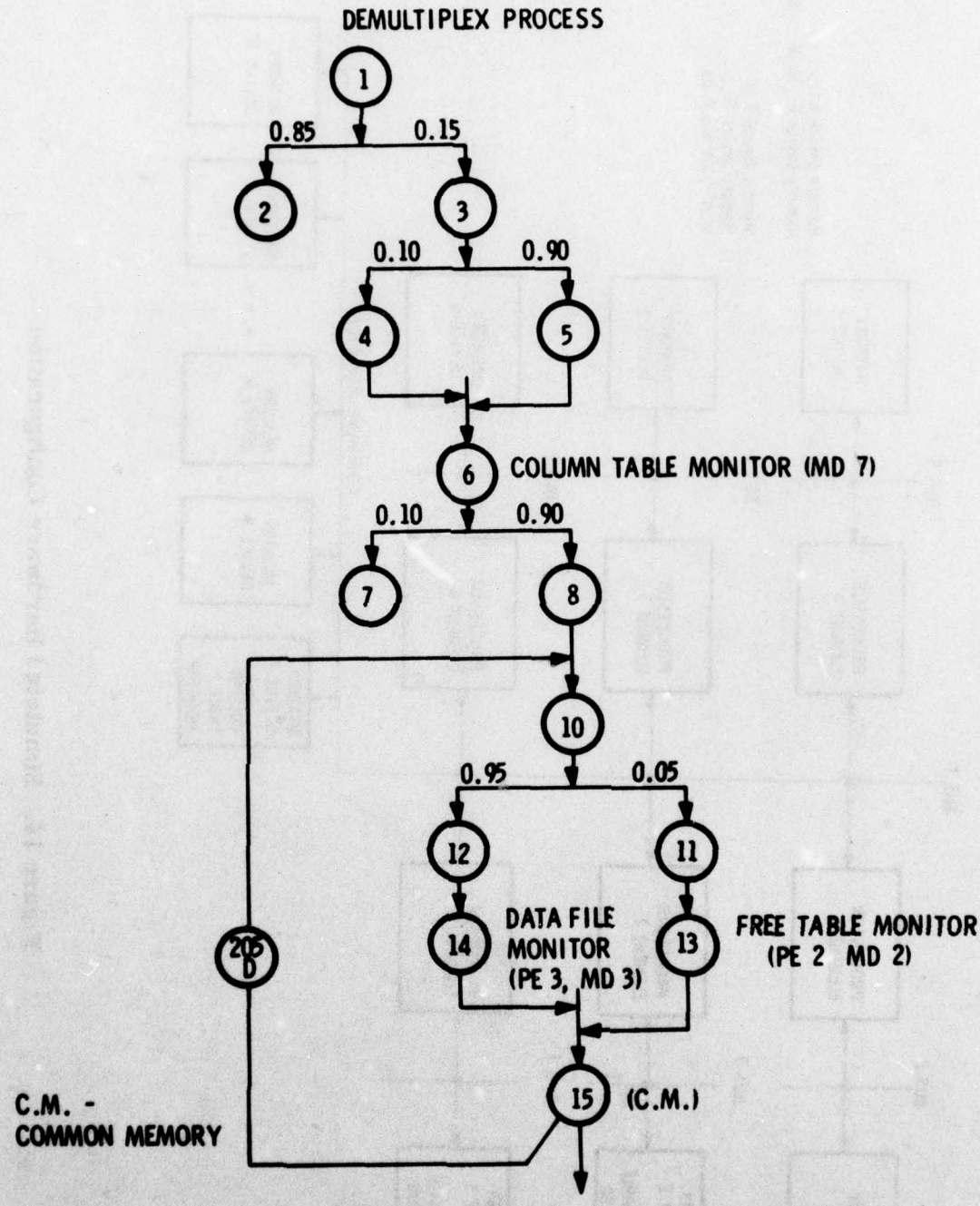


Figure 15. Program Modules for Processing Element 1

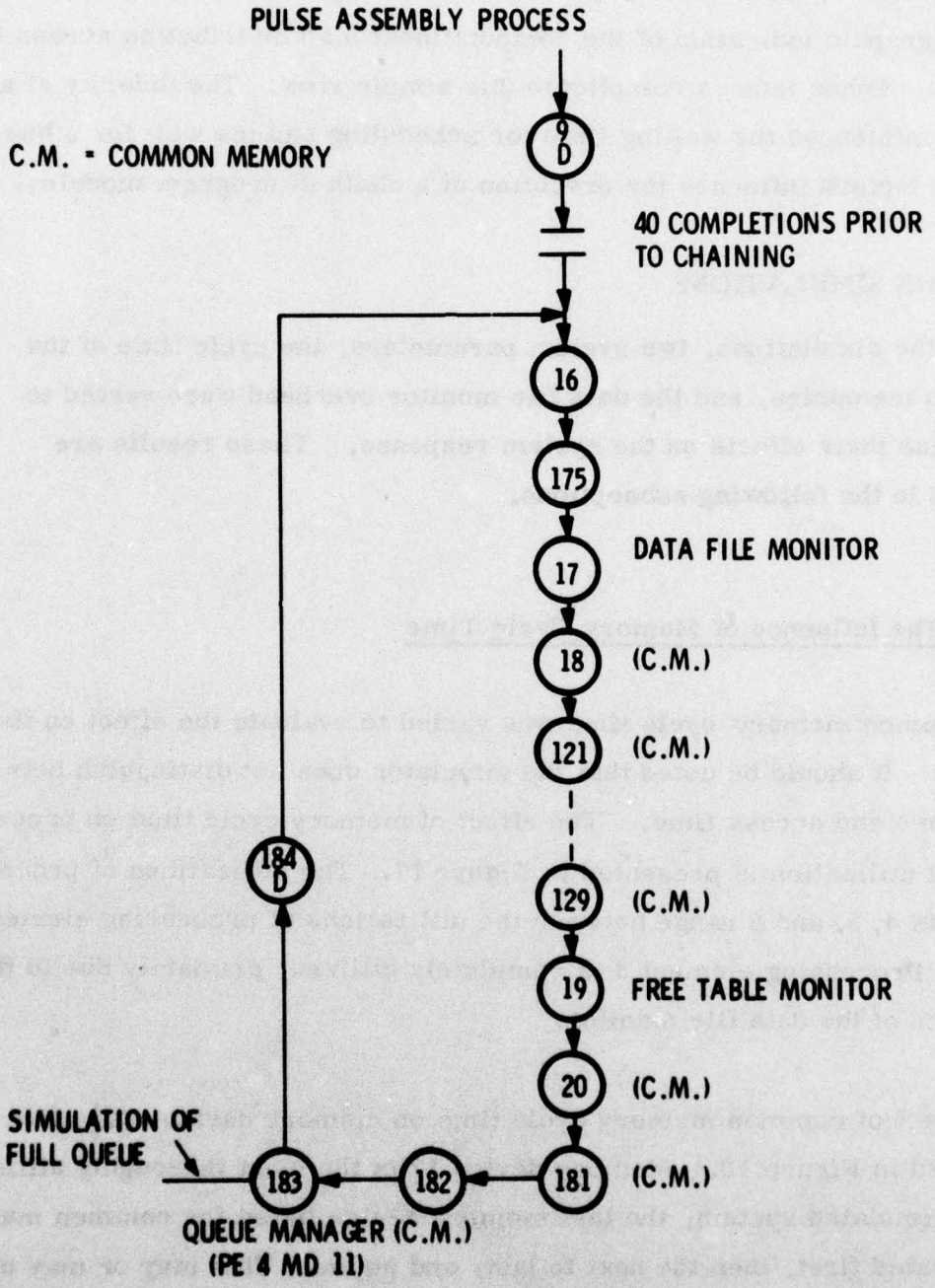


Figure 15. Program Modules for Processing Element 1 (Concluded)

The execution time of each module in PE 1 is depicted in Figure 16. It gives a graphic indication of the computational load distribution across the modules. Other factors complicate this simple view. The priority of a module influences the waiting time for scheduling and the wait for a bus. All of these factors influence the execution of a chain of program modules.

5.3 THE SIMULATIONS

During the simulations, two system parameters, the cycle time of the common memories, and the data file monitor overhead were varied to determine their effects on the system response. These results are covered in the following subsections.

5.3.1 The Influence of Memory Cycle Time

The common memory cycle time was varied to evaluate the effect on the system. It should be noted that the simulator does not distinguish between cycle time and access time. The effect of memory cycle time on processor element utilization is presented in Figure 17. The utilizations of processing elements 4, 5, and 6 range between the utilizations of processing elements 1 and 2. Processing element 3 is completely utilized, primarily due to the presence of the data file monitor.

The effect of common memory cycle time on memory device utilization is indicated in Figure 18. Memory device 17 is the most thoroughly utilized. In the simulated system, the last memory device listed for common memory is allocated first, then the next to last, and so on. That may or may not correspond to an actual implementation. Memory device number three is used more than the other local memories due to the presence of the data file monitor.

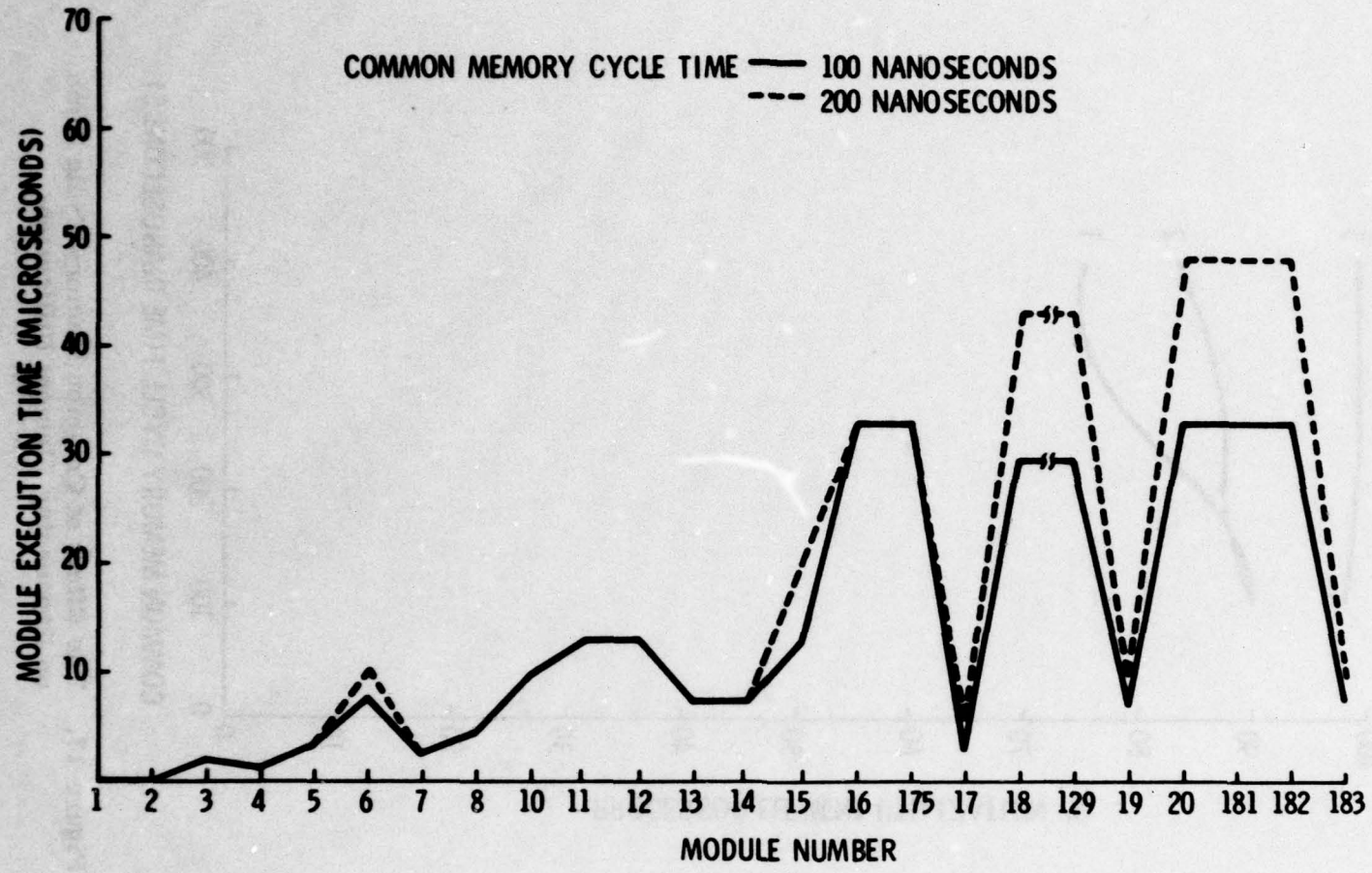


Figure 16. Execution Time of the Program Modules for Processing Element 1

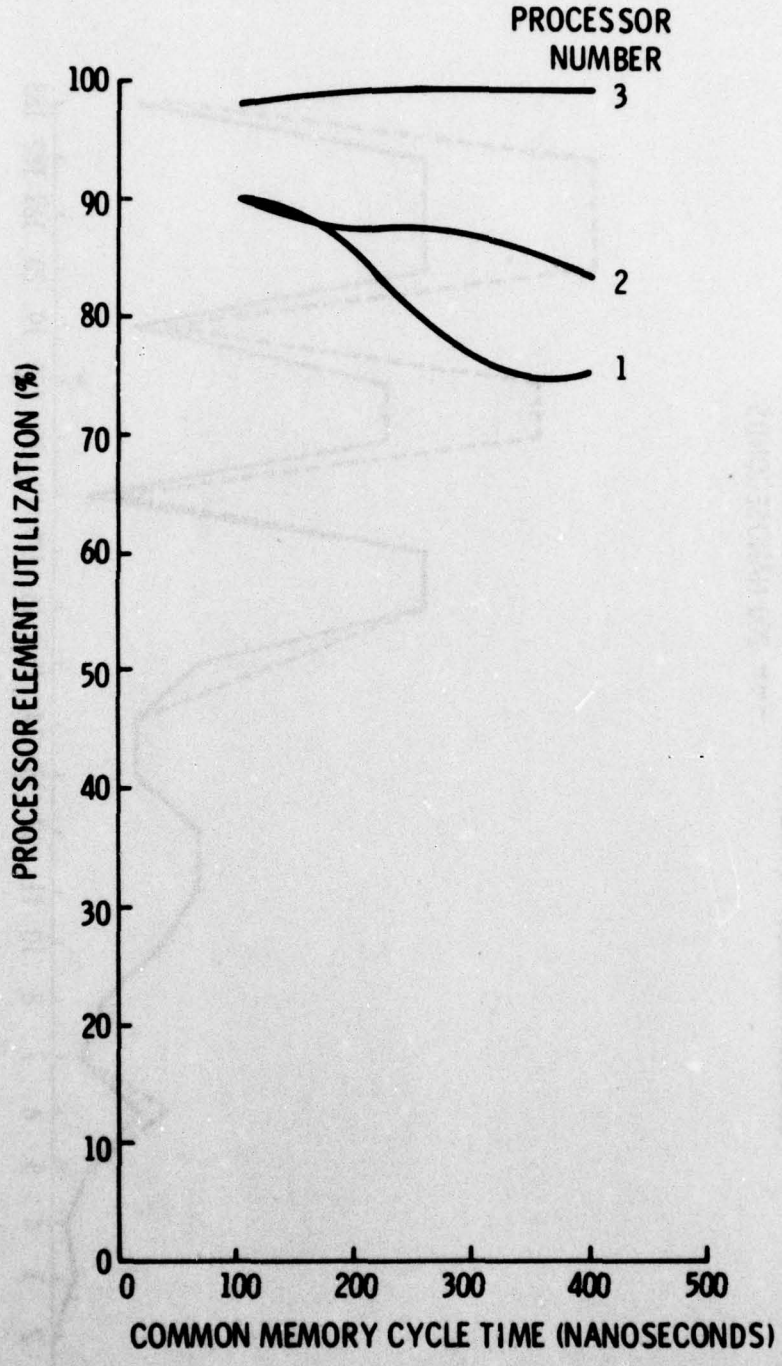


Figure 17. The Effect of Common Memory Cycle Time on Processing Element Utilization

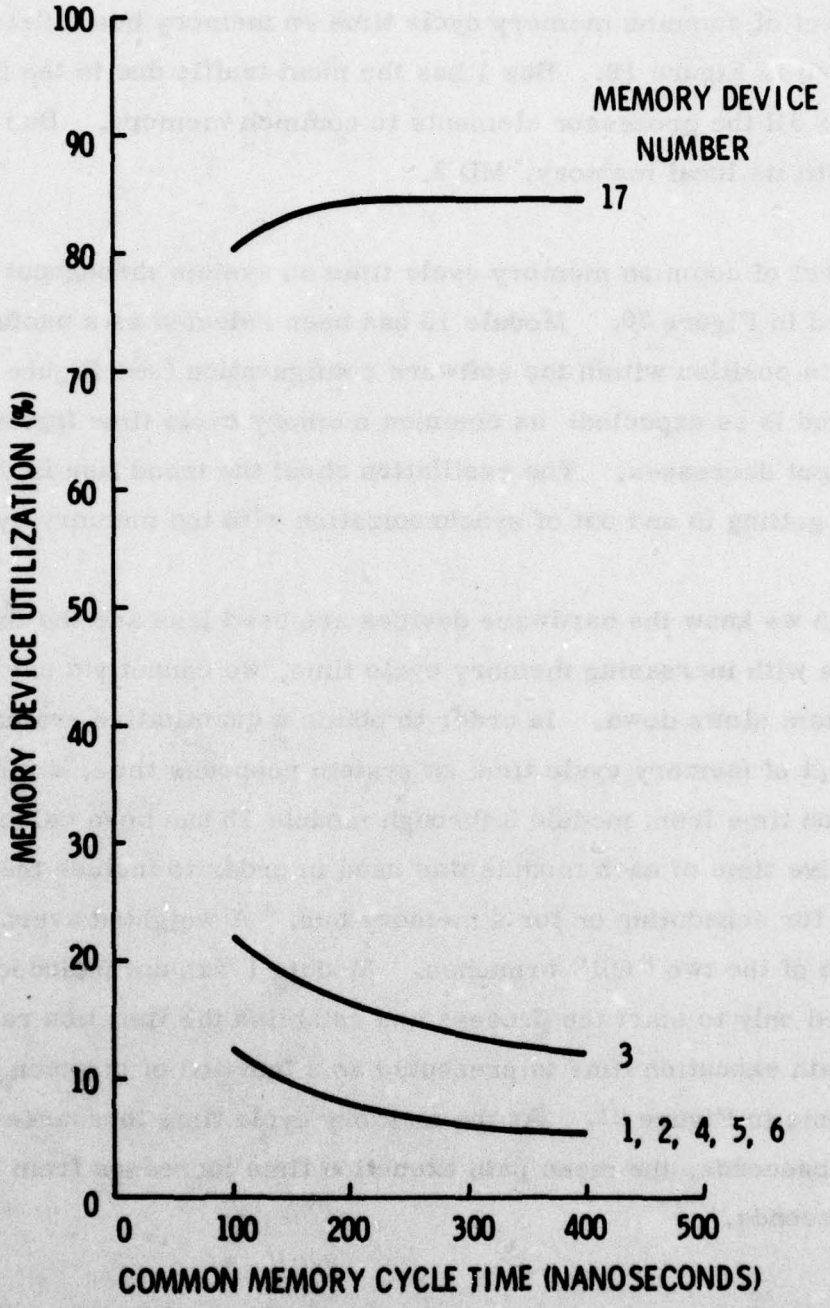


Figure 18. The Effect of Common Memory Cycle Time on Memory Device Utilization

The effect of common memory cycle time on memory bus utilization is presented in Figure 19. Bus 1 has the most traffic due to the fact that it connects all the processor elements to common memory. Bus 4 connects PE 3 with its local memory, MD 3.

The effect of common memory cycle time on system throughput is indicated in Figure 20. Module 15 has been selected as a useful indicator due to its position within the software configuration (see Figure 15). The trend is as expected: as common memory cycle time increases, throughput decreases. The oscillation about the trend line is due to the module getting in and out of synchronization with the memory cycle.

Although we know the hardware devices are used less and the throughput declines with increasing memory cycle time, we cannot yet say how much the system slows down. In order to obtain a quantitative evaluation of the effect of memory cycle time on system response time, a mean path execution time from module 3 through module 15 has been calculated. The active time of each module was used in order to include the time spent waiting for scheduling or for a memory bus. A weighted average was used for each of the two "OR" branches. Module 1 was not included since it was used only to start the process and establish the iteration rate. The mean path execution time is presented as a function of common memory cycle time in Figure 21. As the memory cycle time increases from 100 to 800 nanoseconds, the mean path execution time increases from 130 to 530 microseconds.

The effect of common memory cycle time on bus waiting time, for module 15, is presented in Figure 22.

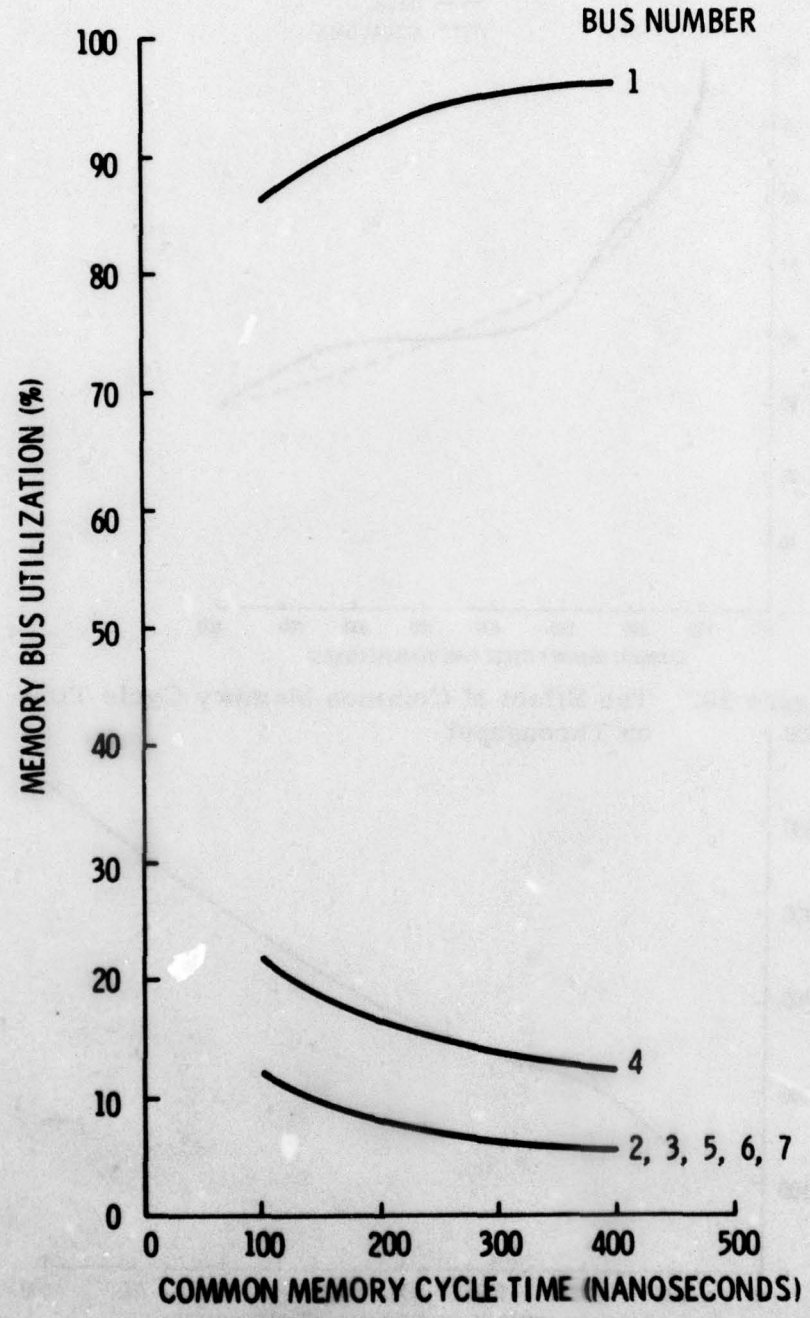


Figure 19. The Effect of Common Memory Cycle Time on Memory Bus Utilization

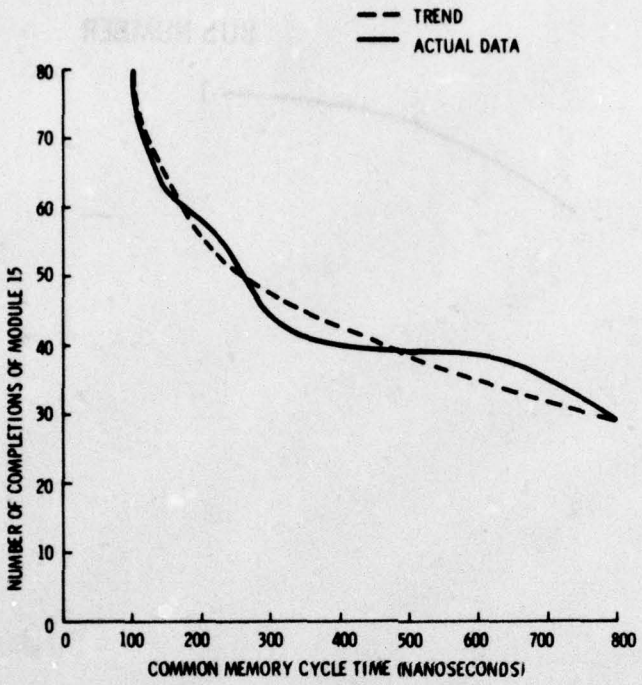


Figure 20. The Effect of Common Memory Cycle Time on Throughput

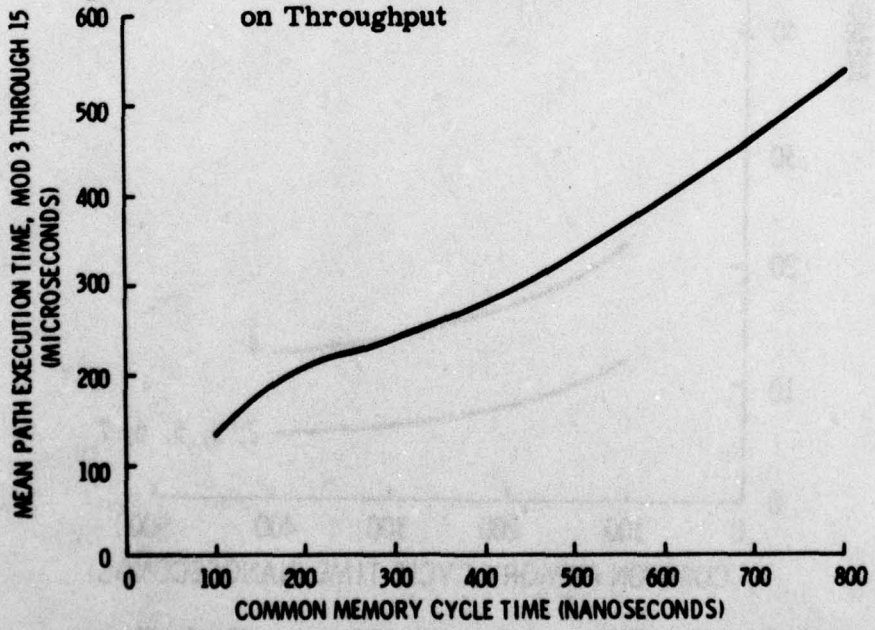


Figure 21. The Effect of Common Memory Cycle Time on System Response Time

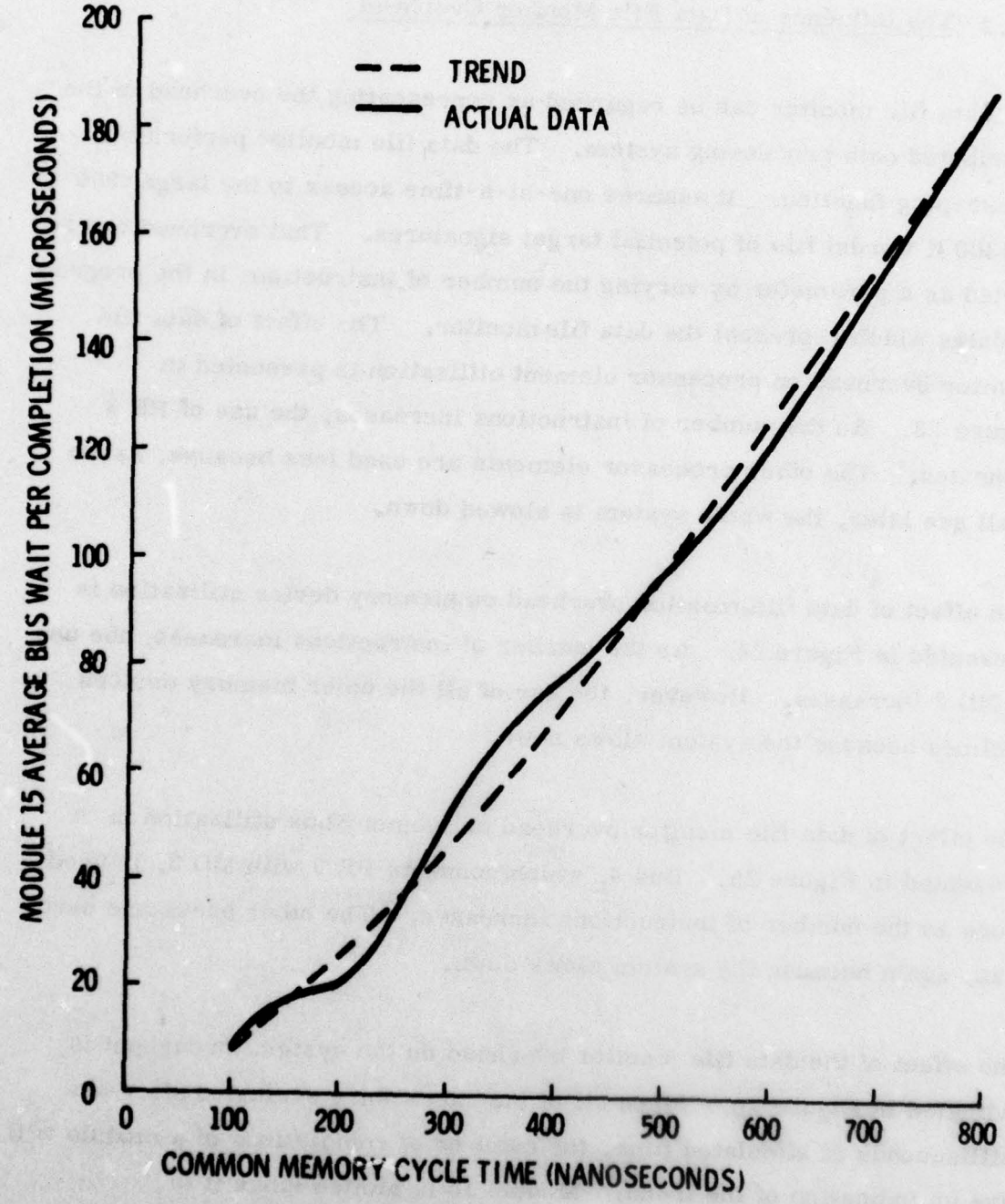


Figure 22. The Effect of Common Memory Cycle Time on Bus Waiting Time

5.3.2 The Influence of Data File Monitor Overhead

The data file monitor can be regarded as representing the overhead of the distributed data processing system. The data file monitor performs a gatekeeping function. It assures one-at-a-time access to the large (600 to 1200 K words) file of potential target signatures. That overhead can be varied as a parameter by varying the number of instructions in the program modules which represent the data file monitor. The effect of data file monitor overhead on processor element utilization is presented in Figure 23. As the number of instructions increases, the use of PE 3 saturates. The other processor elements are used less because, as we shall see later, the whole system is slowed down.

The effect of data file monitor overhead on memory device utilization is presented in Figure 24. As the number of instructions increases, the use of MD 3 increases. However, the use of all the other memory devices declines because the system slows down.

The effect of data file monitor overhead on memory bus utilization is presented in Figure 25. Bus 4, which connects PE 3 with MD 3, is used more as the number of instructions increases. The other buses are used less, again because the system slows down.

The effect of the data file monitor overhead on the system throughput is indicated in Figure 26. Since all of the cases were evaluated after ten milliseconds of simulated time, the numb er of completions of a module will give an indication of the trend. Module 15 is plotted since it is immediately down stream of the first use of the data file monitor. The decrease in throughput with increased overhead is apparent.

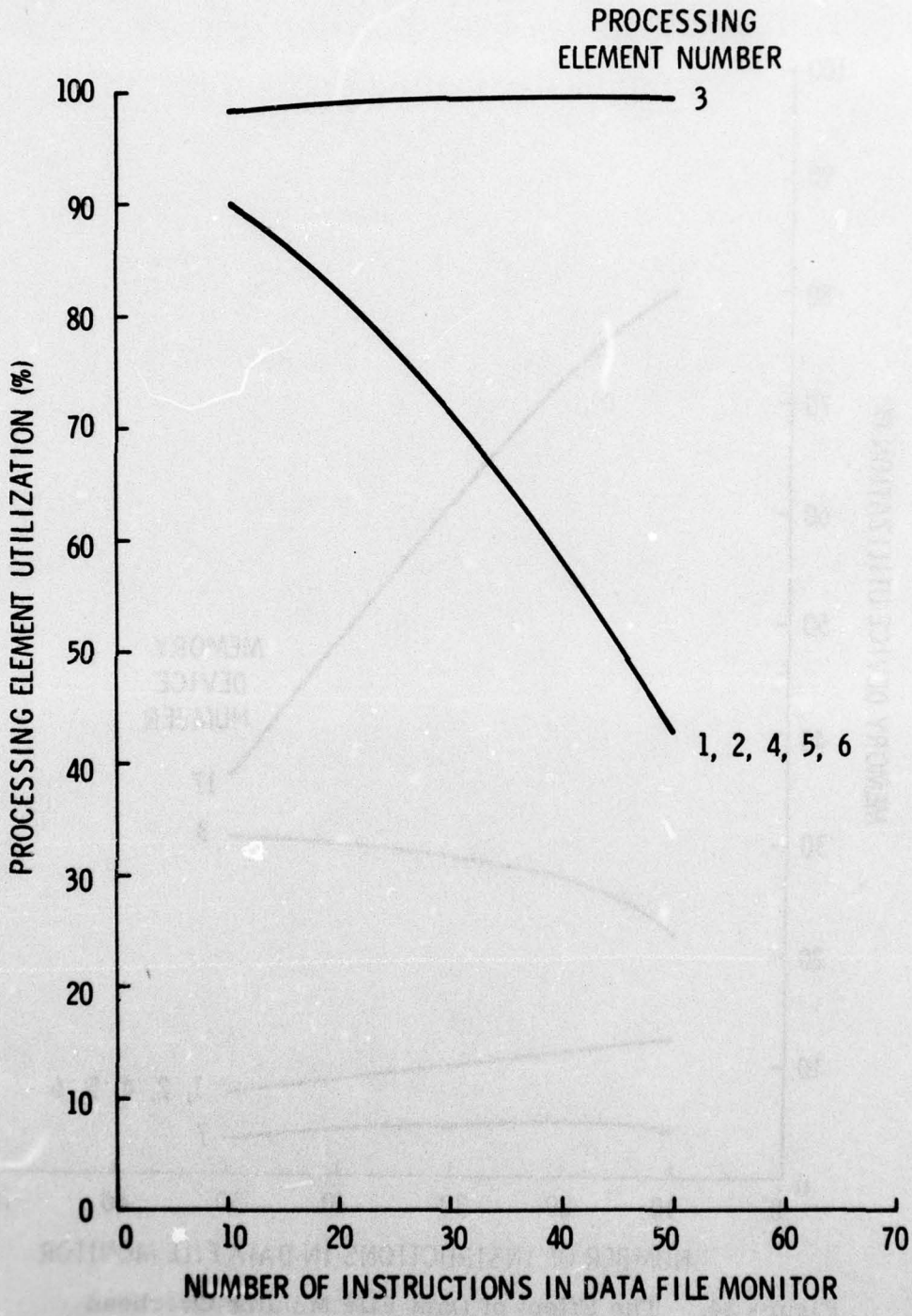


Figure 23. The Effect of Data File Monitor Overhead on Processing Element Utilization

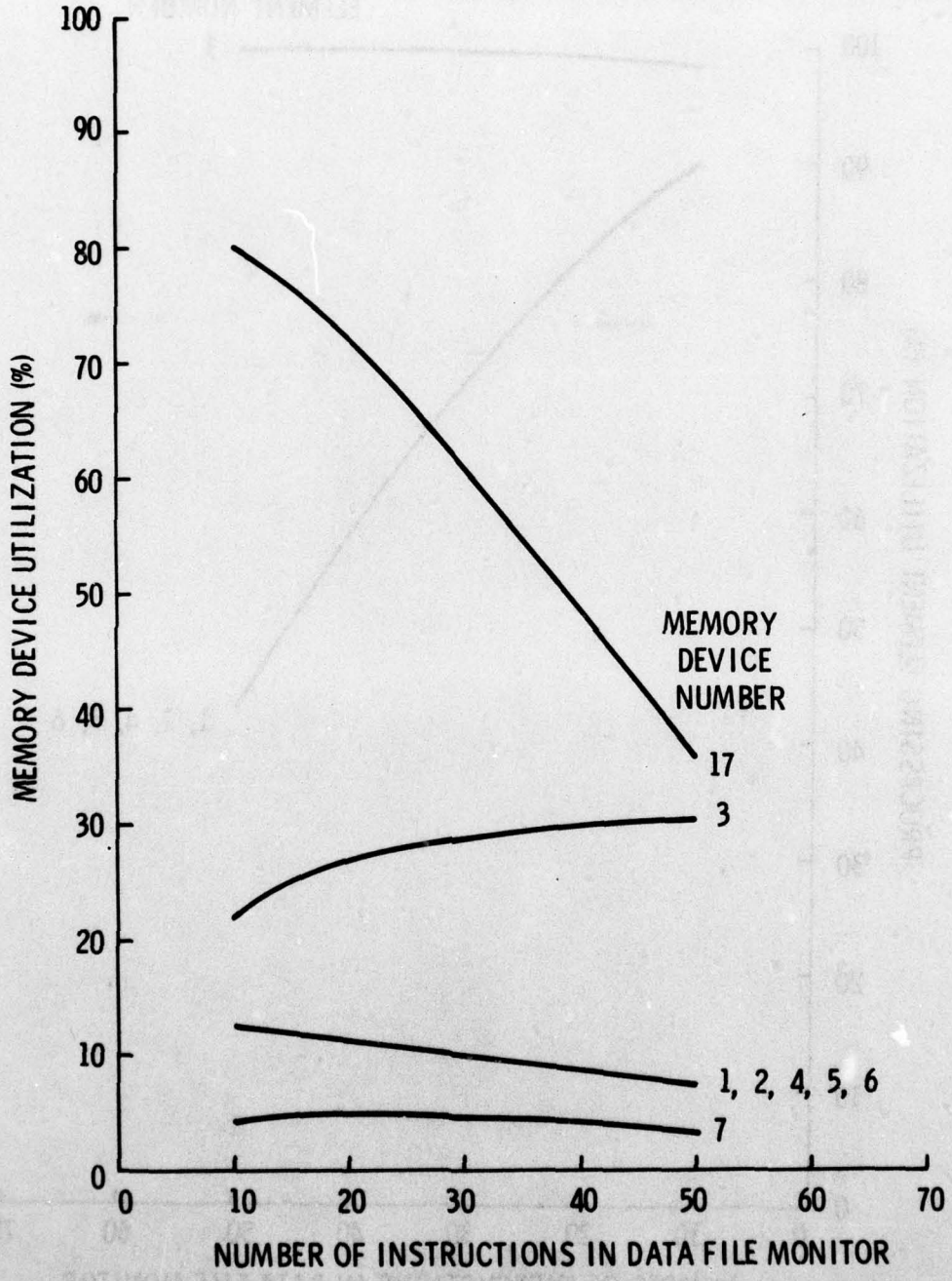


Figure 24. The Effect of Data File Monitor Overhead on Memory Device Utilization

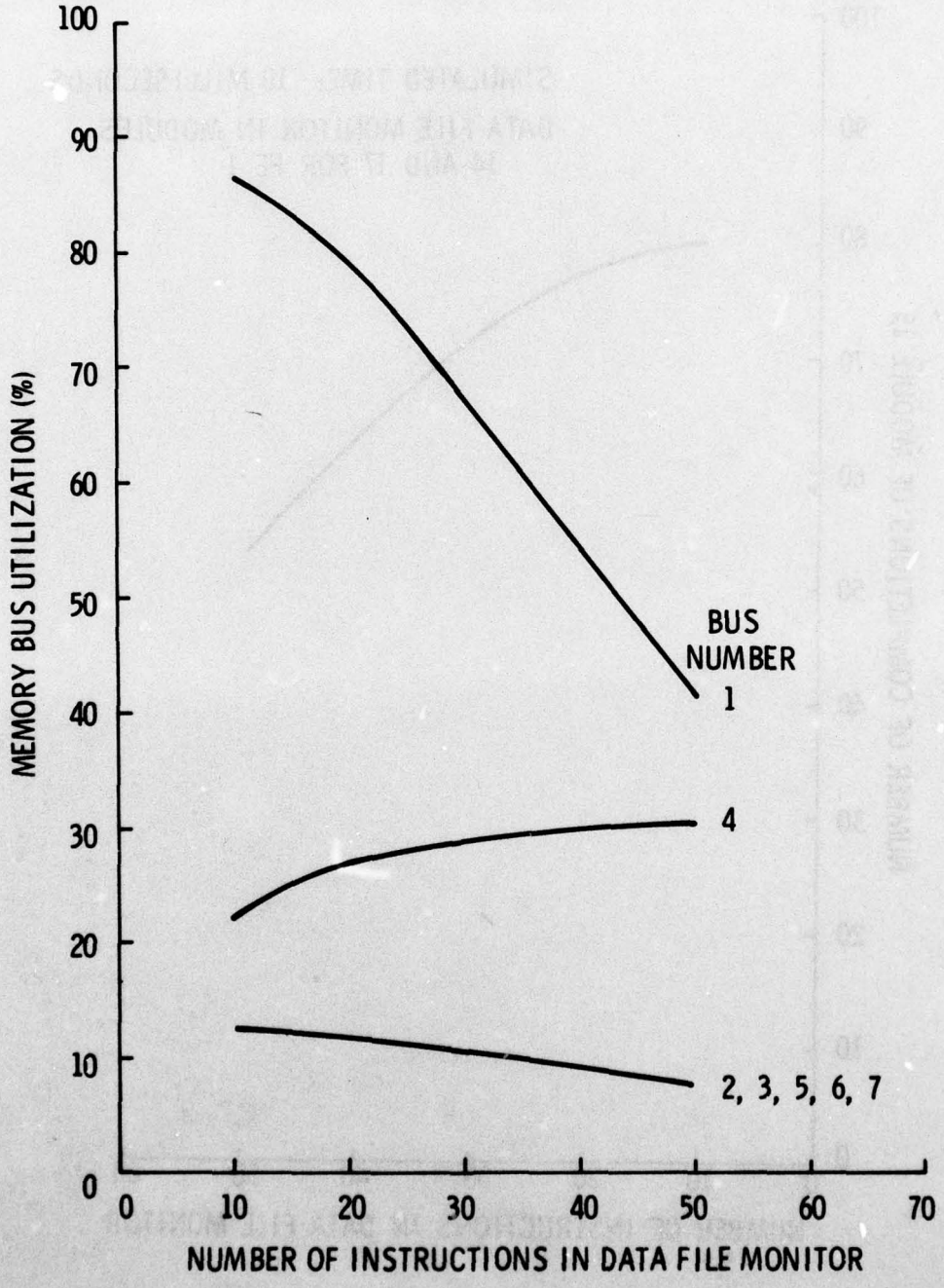


Figure 25. The Effect of Data File Monitor Overhead on Memory Bus Utilization

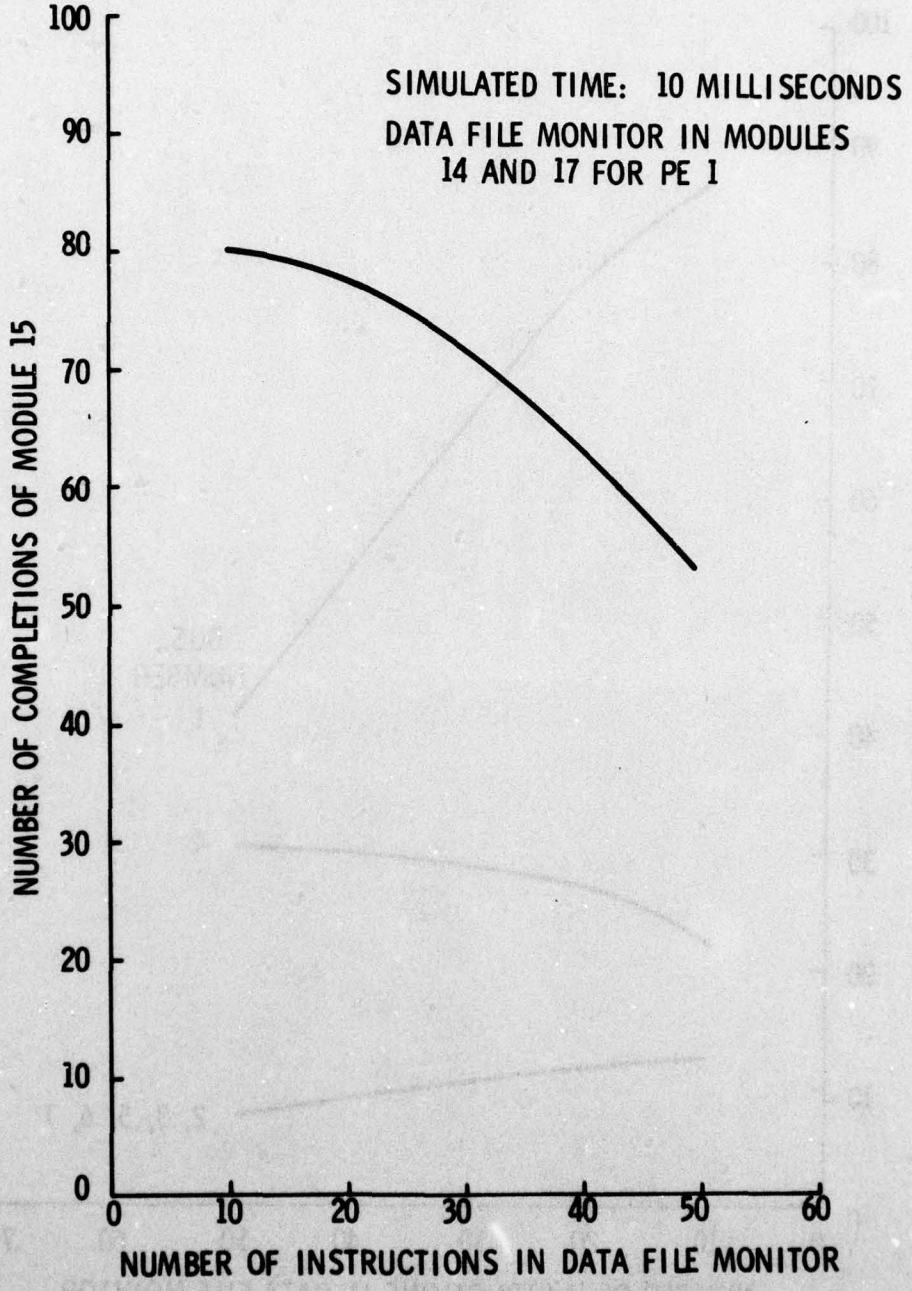


Figure 26. The Effect of Data File Monitor Overhead on Throughput

In order to obtain a quantitative evaluation of the effect of data file monitor overhead on system response time, a mean path execution time from module 3 through module 15 was calculated. The mean path execution time is presented as a function of the number of instructions in the data file monitor, in Figure 27. As the number of instructions increases from 10 to 50, the mean path execution time increases from 130 to 190 microseconds. This gives an indication of how much the system slows down.

5.4 SUMMARY

It is possible to gain some useful insight from the simulator. However, the preparation of input can be an elaborate process. A great deal of effort was expended on MMBC in order to describe the functions required of the computer for the mission, a careful examination of algorithms was undertaken in order to estimate the computational loading, an architecture was designed, and finally the information was mapped into the format required by the simulator. Because of H6080 memory requirements, we simulated only six of 18 microprocessors for MMBC. Two hundred and ten software modules were used for the six PE version. The cost of running a case in daytime would range from $25 to $100. Of course, that would be cheap when compared to using trial-and-error hardware experiments for design.

This experience elicited the following suggestions for modifications to the simulator:

1) The inclusion of a queueing module for the explicit representation of a queue.

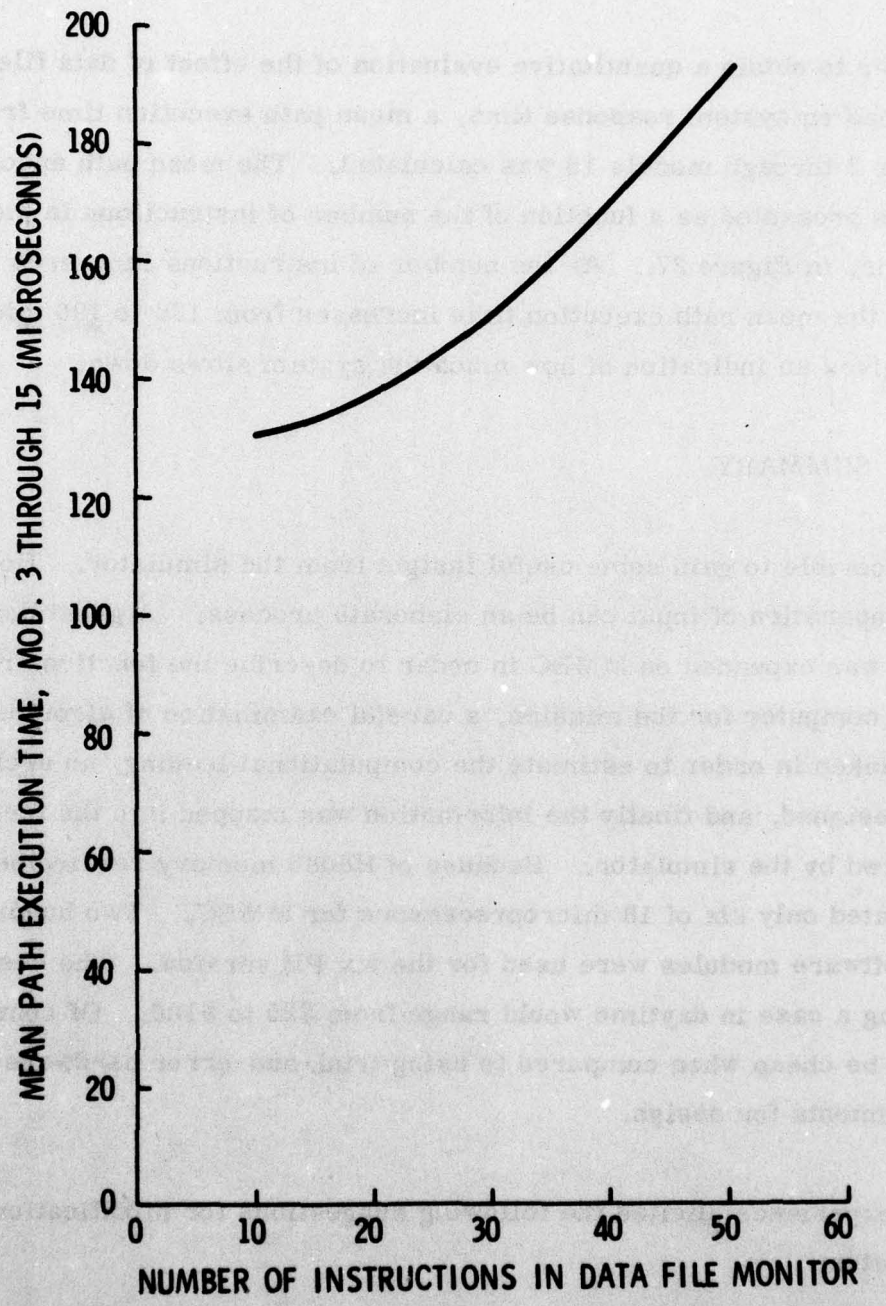


Figure 27. The Effect of Data File Monitor Overhead on System Response Time

2) More flexible selection of a memory device for writing in common memory (random or round-robin).

3) System conversion for use on the CDC 7600 at the ARC.

4) Correction of the automatic memory allocation algorithm.

5) Provision for an indication of system response time, such as the mean path execution time and the variance (or standard deviation). We were surprised to learn that this capability was not already in the simulator.

It would also be convenient for the user if an explicit numbering of the modules (rows in MCHAN) were included. If multiple copies of the input array are employed in the current system, then one may be able to reduce the memory requirements by reprogramming.

The simulator can be used to evaluate alternative computer architectures. It is a flexible facility that allows designers to specify both hardware and software architectures, and the mapping of the software functionality onto the hardware modules. It should be used to evaluate DDP architectures proposed for BMD systems.

SECTION 6

PLANS FOR EXPERIMENTAL FACILITIES

Many proposed experiments require data processing systems. Some experiments use data processing to simulate designs, in which case any general-purpose system supporting a convenient simulation language will be adequate. Other experiments use emulations or very complex simulations combined with emulations, in which case a specially-configured DDP emulation facility is needed.

In Subsection 6.1, we discuss first the requirements for an emulation/prototype facility and propose a specific configuration that could be added to the ARC facility in Huntsville. Systems Development Corporation has independently proposed a configuration for the ARC; in Subsection 6.2, we summarize their configuration and present some reactions to that design.

6.1 FACILITY PROPOSAL BY HONEYWELL

Our facility proposal covers both hardware (Subsection 6.1.1) and software needed (Subsection 6.1.2) to support the machine-based experiments. Mechanism experiments generally require simulating or emulating DDP architectures, or architectural subsystems measuring their performance.

The literature describes numerous simulation/emulation based attempts to resolve many of these same issues. It is clear that this approach, when implemented on a large-scale, general-purpose computer, is both difficult

and expensive. DDP systems are often asynchronous collections of independent modules competing for the system resources. Thus, a simulation must provide a fine time base granularity so that the effects of module interdependence (as they contend for system resources) can be measured adequately. As the detail and fidelity of the experiments increases, the system requirements increase dramatically.

As a result of these considerations and the analyses performed on other BMD contracts,[21] it is clear that for a practical DDP architectural experiment to be feasible, it will be necessary for the experimental facility to resemble the logical and physical structure of the target architecture. Since a wide variety of architectures is being postulated, the experiment host must be very general and flexible. In addition, the experiment host must be capable of sufficiently high performance in such a manner that the entire experiment can be run to completion within a reasonable period of time. In any case, this time period must be well within the expected MTBF of the total system.

6.1.1 Hardware Requirements

The following equipment is sufficient to perform the suggested experiments. We assume that ARC's CDC 7600 is available; we plan to make maximum practical use of this existing system. The most significant role for this system will be statistical data reduction and reasonable presentation of the experimental results.

The minimum additional hardware required in the ARC is:

- a set of small identical general purpose processors (called the "small computers" in this document),
- an adequate amount of local memory associated with each small processor,
- a single processor augmented with a sufficient quantity of additional resources to support extensive software development for all small processors (called the "medium computer" in this document),
- several (block-transfer-oriented) mass storage devices,
- a set of terminal I/O devices,
- versatile computer-to-computer interconnection structures, i.e., DMA channels
  modems and controllers
  CDC 7600 interface, and
- performance analyzer/monitor equipment.

A discussion of their individual characteristics, uses, and an example meeting all the explicit and subjective system requirements are presented in the following subsections. The examples should further clarify the system needs. The actual hardware must meet or exceed the capability of the examples in all respects.

6.1.1.1 Small Processors--A set of identical, small, general-purpose processors is required to efficiently emulate several concurrent asynchronous tasks. An adequate number of processors--at least four--will be needed to demonstrate that a DDP system can function in nontrivial, nondegenerate cases. These processors should be identical to avoid any duplication of support equipment and software development tools. A small mini/micro computer would be adequate. Word lengths may be as small as 16 bits if multiple precision arithmetic is available.

Adequate processor throughput will be approximately 200K ops; the DEC PDP11/03, for example, seems to be an adequate processor.

6.1.1.2 Memory--Central memory sufficient to avoid program swapping should be associated with each small computer. The PDP11/03 normally includes 4K words RAM to which should be added a 16K RAM increment for a total of 20K words.

6.1.1.3 Medium Processor--The medium processor will be used for developing software for the small processors, supervising the experiment, communicating with the CDC 7600, and for controlling communications among the small processors (see Subsection 6.1.1.6). The software development function demands the most hardware support.

To support editors, file management, and human users, moderate processing speed should be coupled with a line printer, a medium-speed terminal, and some mass storage. Our estimates show that a system with 32K words of central memory, five million words of disc, and a 300 LPM line printer would be adequate.

The medium processor should be software compatible with the small processors to obviate the need for cross-assemblers. Furthermore, DMA access from the medium processor to the small processors will facilitate loading code and data into the small processors; this facility can also be used to manage communications (see Subsection 6.1.1.6).

As an example, suppose that PDP-11 processors are used; then the following configuration would be adequate:

- PDP11/34 processor
- 32K words RAM
- LA36 Decwriter II 30 cps terminal
- Memory management option
- RK11/RK05 Disk System (5M words storage on line)
- Real-time clock
- 300 LPM line printer
- DMA ports to all small processors
- Port to 7600 PP

6.1.1.4 Mass Storage--Hardware to support distributed data bases must be included in the experimental facility to assist experiments studying data base management in distributed systems (see Subsection 3.2.1). Each small computer should have a small mass storage device, such as a floppy disk system. For example, with a DEC PDP 11 system, the RXV11 dual floppy drive providing 512K bytes of storage should be adequate.

6.1.1.5 Terminals--Simple, irrefutable demonstrations of interprocessor interactions can use terminals attached to each of the small computers. Modern terminals with a speed of at least 30 cps would be sufficient; CRT displays are especially attractive. Furthermore, if each small computer had such a terminal, it could be used by itself to perform useful work and to debug experiment software.

6.1.1.6 Interprocessor Communications Equipment--The interconnection facility providing communications among a collection of computers is the heart of any DDP system. The experiments being developed under this contract require a wide variety of interconnections. DMA channels between each of the small computers and the medium computer running the software development system will provide the means of transferring software to each computer and will also provide an interconnection that is versatile enough for many applications.

Communications paths among the small computers can be emulated using the DMA paths through the medium computer, but direct communications would be more graphic, would provide more realistic emulation, and would be easier to interface to. Therefore, we propose that each small computer have at least two communications interfaces. The maximum flexibility would be provided if these interfaces could be interconnected directly or through buses.

The configuration in Figure 28, based on PDP-11 equipment, illustrates the use of DMA devices to communicate with the medium computer. This interconnection scheme has an advantage in that transfers can take place among LSI-11's independent of the PDP11/34. This system could implement

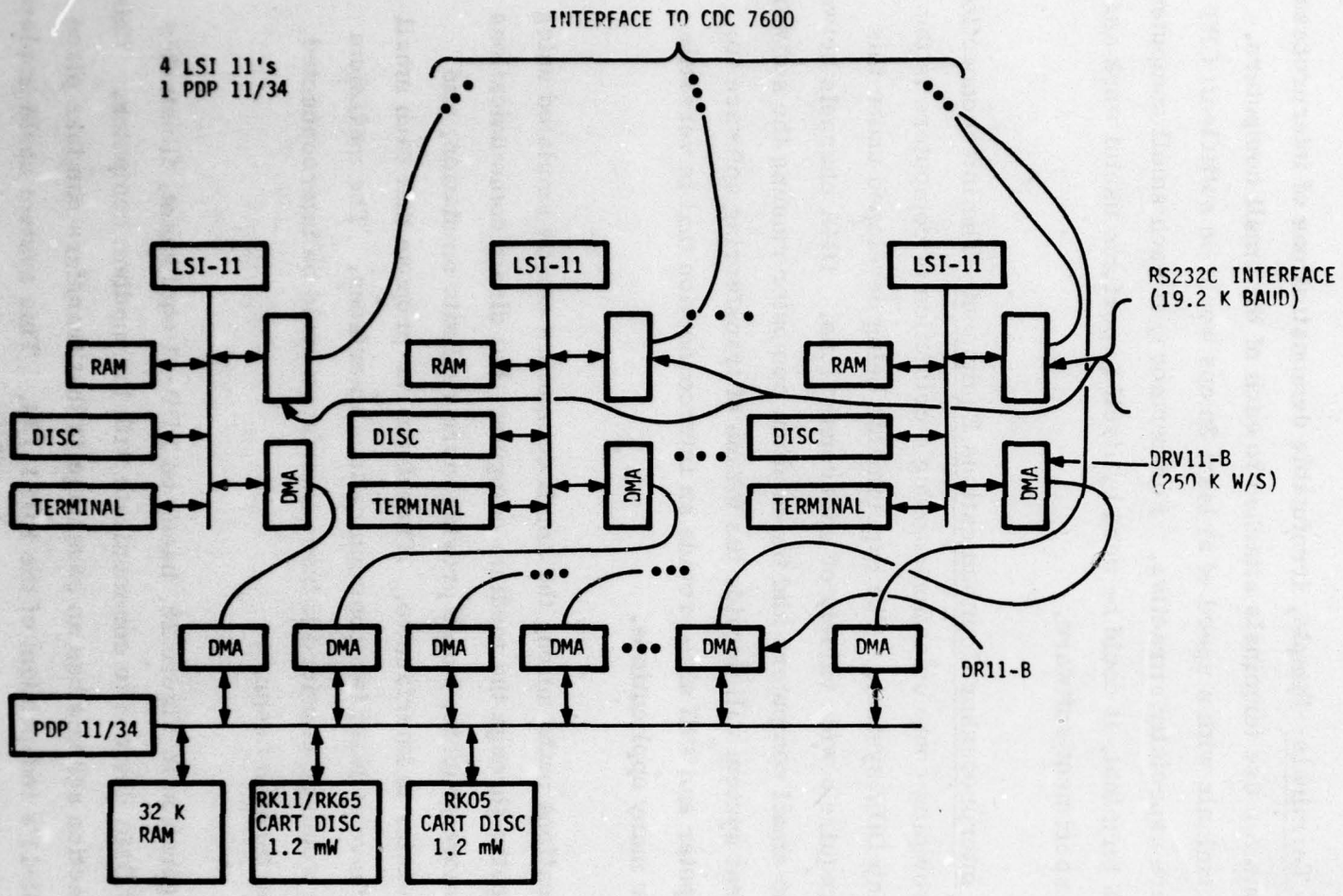


Figure 28. Proposed Experiment Host

AD-A047 478 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
DISTRIBUTED DATA PROCESSING TECHNOLOGY. VOLUME VII. APPLICATION--ETC(U)
SEP 77 R G ARNOLD, W E BOEBERT, E D JENSEN DAS660-76-C-0087

UNCLASSIFIED 77SRC71 NL

2 OF 2
AD
A047478

END
DATE
FILMED
1 -78
DDC

a general bus structure and protocol at a transfer rate of about 250K words/second. The structure and protocols of other interconnection schemes could be implemented by having the PDP11/34 take control of every transfer, passing data among computers according to the program information available to it. Thus, the system could mimic loops, stars, buses, etc. Transfers along "nonexistent" simulated links would be prevented by the PDP11/34's software.

A severe data rate penalty must be paid when interaction with the medium computer is required to mimic nonexistent interconnection structures.

Because these structures are important in DDP experiments and because we must maintain reasonable real-time ratios, additional capability must be included to explicitly connect the small computers to each other. High bandwidth interconnections can be provided through the bus interface communications. However, geographically distributed systems may have low bandwidth interconnections that could be emulated through asynchronous serial communications adapters and modems. These devices will allow connections among the small computers and to outside facilities such as ARPANET; they are not shown in Figure 28 to simplify the presentation.

Interconnection hardware must connect the experiment to the CDC 7600 system. It is our understanding that a PPU port will be used to connect with the 7600. This will require a DMA channel to the medium computer and a channel-to-channel adapter. For those experiments requiring a large communication bandwidth to the CDC 7600 (either simulating stimuli or gathering performance monitoring data), each small computer may require a data channel to the 7600, as shown in Figure 28.

6.1.1.7 Performance Analyzers/Performance Monitors--Hardware probes for performance monitoring may be required to eliminate any perturbations that might be caused by other measuring mechanisms. It appears that the DMA communication from the medium computer to the small computers will be sufficient for performance monitoring when the medium computer is not needed to emulate nonexistent interprocessor communication structures. Thus, hardware probes may be necessary when the medium computer plays an active role in the experiment.

6.1.2 Software Requirements

Software for the experimental facility must support both the performance of the experiment and the collection of the experimental results. Different software will be required for the different classes of processors within the system. This document will summarize the support software needed for each processor class; software specific to each experiment must be developed with the experiment. Some of this specific software has been discussed in previous sections where the experiments are described.

Details of the software module interfaces remain to be defined in future development studies; these details must be developed after the hardware configuration is selected so that field sizes can be determined.

6.1.2.1 Small Processor Software--Most of the code executed on the small processors is specific to individual experiments. Three classes of support software will be needed:

- Communications interface drivers
- Communications network drivers
- Mass memory drivers

6.1.2.1.1 Communications Interface Drivers--The communications interface drivers will support the details required to initiate and monitor message transmissions between small processors. These software modules provide details necessary to initiate transfers, but they do not perform routing functions (see Subsection 6.1.2.1.2). Separate modules must support the serial asynchronous interfaces and the bus system interconnecting the small processors.

6.1.2.1.2 Communications Network Drivers--The communications networks interconnecting distributed processors will be the subject of some experiments (see Subsection 3.2.4 and 3.2.5). Other experiments not directly studying this function will require communications support. The software driving these communications links should accept message transmission requests, including destination processor number and the message to be transmitted. They will establish a status word that may be queried by the experiment software to determine whether the message has been received and acknowledged. The module will automatically retry transmission a number of times that may be controlled by the experiment software. (No retries shall be an allowed possibility.)

Furthermore, the communications drivers will include facilities to collect timing data as needed. This requirement needs further study. The intention is to measure communications overhead so that any overhead that

is a consequence of a particular interconnection scheme can be factored from the experimental data if the purpose of the experiment is to measure other aspects of system performance.

Tables accessed by the communications network drivers shall define the interconnection topology; the drivers will use these tables to route messages through the system. Optimum routing is not required.

6.1.2.1.3 Mass Memory Drivers--The mass memory driver must provide minimal facilities to allow the mass memory to be viewed as a file system. Access control and protection need not be provided, however, as these functions will be subject to experimentation (see Subsection 3.2.1). Variable file sizes will be supported by this software.

6.1.2.2 Medium Processor Software--The medium processor supports software development for the small processors (and for itself), experiment functioning, and experiment result monitoring. Generic software for each of these functions should support the experiment facility.

6.1.2.2.1 Software Development Aids--The medium processor will provide standard software development aids including compilers, assemblers, editors, and a file system. If the small processors are compatible with the medium processor, the software development system provided by the hardware vendor should be sufficient. If, however, the small processors are not compatible, cross-assemblers and cross-compilers may have to be developed. We hope that almost all experiment-dependent programming can use a high-level language. The execution efficiency is not paramount; fast development times are much more important for performing many experiments in a short period of time.

Nonstandard software to copy the code into the small computers via DMA access to those machines will be required. These programs should allow the development engineer to specify which code is to be loaded into which machine. Usually, the same code will be loaded into each small processor, with individual parameters loaded as data.

6.1.2.2.2 Experiment Aids--One set of software modules will be required to support the interprocessor communication among the small processors if the medium processor must become involved in such transactions (see Subsection 6.1.1.6). If, however, the small processors handle their own intercommunication (as described in Subsection 6.1.2.1.2), this medium processor software is not required.

Another set of software modules will be required for real-time performance monitoring during the experiment.

Communications Software

This software, required if the small processors are interconnected only through the medium processor (see Subsection 6.1.1.6), performs message switching functions to emulate arbitrary interconnections of the small processors. Tables describing the actual interconnections and the desired interconnections would be consulted to determine how to transmit each message.

Performance Monitoring Software

The medium processor will monitor the performance of the experimental system by examining the status of small processors as necessary. All experiment-dependent software executing in the

small processors will maintain, in common memory locations, all status information required for performance monitoring. The medium processor shall examine these status words according to a predetermined (table-driven) scheme.

This software design must be coordinated with the design of the data reduction software for the 7600. The medium processor may have sufficient capacity to compute simple performance statistics before relaying them to the 7600. This interface has not been studied in detail.

6.1.2.2.3 Interface to the CDC 7600--The CDC 7600 interface software that runs in the medium processor shall coordinate the communications between the medium processor and the CDC 7600. The CDC 7600 controls this communication since it exercises DMA access to the medium processor. The interface software in the medium processor must leave control messages for the CDC 7600 in predetermined locations where the 7600 will find them.

The 7600 also provides inputs to the experiment from the SETS system. These inputs are collected by the medium processor and disseminated (in an experiment-dependent manner) to the small processors if appropriate. Message dissemination is not a function of the standard interface software.

6.1.2.3 Software for the CDC 7600--The CDC 7600 not only collects, processes, and presents experiment performance data, but it also sends signals from the SETS system to the experiment hardware. All of this information must pass through the same 7600PP-medium processor DMA

channel. The 7600PP must have software modules that monitor these messages, using control blocks placed in the medium processor's memory. Three classes of software are required:

1. SETS interface software
2. Medium processor interface software
3. Statistics reduction and presentation software

Further definition is required in each class.

6.2 RESPONSE TO FACILITY PROPOSAL BY SDC

We first summarize the SDC proposal (in Subsection 6.2.1) to provide context for our response (in Subsection 6.2.2).

6.2.1 Facility Proposal by SDC

Figure 29 shows a hardware configuration proposed by System Development Corporation[22] to be added to the ARC system in support of DDP experiments. Since no suggestions for support software to aid these experiments have been received by Honeywell, we have evaluated the hardware configuration alone.

6.2.2 Honeywell Response to SDC Facility Proposal

Generally, Honeywell feels that the proposed facility would be adequate to support the experiments described in this document. However, we do think that the system includes many capabilities and features that we would not need or use to perform experiments. There is one configuration deficiency.

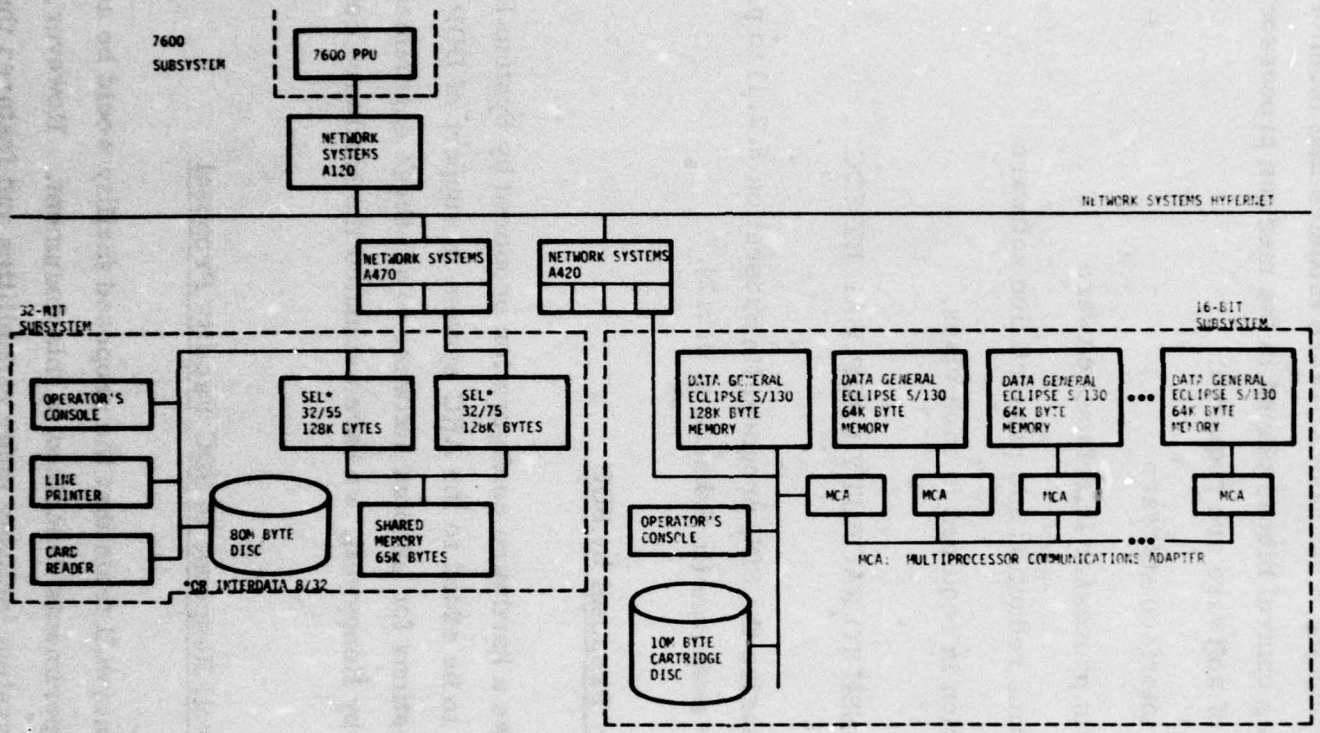


Figure 29. SDC Preliminary Proposal for Testbed Configuration

The deficiency is that the small computers do not have enough capability to communicate among themselves. For additional interconnection flexibility we would add at least two serial modem interfaces and one MCA interface to each small computer. The MCA interfaces would support multi-bus interconnection experiments (see Subsection 3.2.5). Modems would support thin-wire interconnection experiments.

If we were designing the configuration, we would have done the following things differently:

1. We would not have included any 32-bit system. Our experiments do not require this capacity; we would rather spend the money on additional 16-bit machines so that we can emulate larger networks. [illegible]ng us added confidence that the network simplic[illegible]t a factor in the experimental results.

2. We would have connected the software development peripherals to the larger 16-bit machine rather than to the 32-bit machine, so that software development for the smaller 16-bit machines will not require cross-compilers or cross-assemblers. In fact, standard software development software could be used for the smaller machines.

3. We would have reduced the memory sizes of the small 16-bit machines and used the money to purchase additional machines. We do not think we will need all the indicated space for performing our experiments.

4. We would have chosen PDP-11's for the 16-bit machines. More software is available, and the Carnegie-Mellon k.map interconnections could be used in interconnection topology experiments.

REFERENCES

1. Brinch Hansen, P., Operating System Principles, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.

2. Bhandarkar, D. P., "Analysis of Memory Interference in Multiprocessors," IEEE Trans., C-24, 9, September 1975, pp. 897-908.

3. Thomas, R., discussions at Brown University Workshop on Distributed Processing, 1976.

4. Farber, D.J. and Larson, K. C., "The System Architecture of the Distributed Computer System--The Communications System," Proceedings of the Symposium on Computer-Communications Networks and Teletraffic, Brooklyn, N.Y.: Polytechnic Press, April 1972.

5. Farber, D.J. "The Distributed Computing System," Proceedings of the Seventh IEEE Compcon, February 1973, pp. 31-34.

6. LeLann, G., "Distributed Systems--Towards a Formal Approach," Information Processing 77, North-Holland, 1977, pp. 155-160.

7. Stone, H.S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Trans. SE-3, 1, January 1977, pp. 85-93.

8. Reames, C. C. and Liu, M. T., "Design and Simulation of the Distributed Loop Computer Network (DLCN)," Proceedings of the Third Annual Symposium on Computer Architecture, IEEE Document 75CH1043-5C, January 1976.

9. Reames and Liu, op. cit.

10. Pierce, J. R., "Network for Block Switching of Data", Bell Syst. Tech. J., Vol. 51, July/August 1972, pp. 1133-1143
Pierce, J. R., "How Far Can Data Loops Go?" IEEE Trans. on Comm., COM-20, 3, June 1972, pp. 527-530.

11. Farmer, W. D. and Newhall, E. E., "An Experimental Distributed Switching System to Handle High-Speed Aperiodic Computer Traffic," Proceedings of the ACM Symposium Problems in the Optimization of Data Communication Systems, October 1969.

12. Reames and Liu, op. cit.

13. Jensen, E. D., "A Distributed Function Computer for Real-Time Control," Proceedings of the 2nd Annual Symposium on Computer Architecture, IEEE Publ. No. 75CH0916-7C, January 1975, pp. 176-182.

14. Kinney, L. L., "Analysis of a Multiprocessor System with a Shared Bus," Honeywell Internal Memo to R. G. Arnold, July 13, 1977.

15. Kleinrock, L. and Tobagi, F. A., "Packet Switching in Radio Channels: Part 1--Carrier Sense Multiple-Access Modes and Their Throughput - Delay Characteristics," IEEE Trans. on Comm., COM-23, 12, December 1975, pp. 1400-1416.

16. Jensen, op. cit.

17. Kleinrock and Tobagi, op. cit.

18. Jensen, op. cit.

19. Arnold, F. G. and Schlicht, R. A., "Modular Missile Borne Computers, Functional Simulator for On-Board D. P. Performance Evaluation," prepared by Honeywell, Systems and Research Center for BMDATC under Contract DASG60-76-C-0049, December 1976.

20. Berg, R. O., et. al, "Modular Missile Borne Computers, Final Report, Volume 5, Data Processing Requirements," prepared by Honeywell, Systems and Research Center for BMDATC under Contract DASG60-76-C-0049, November 1976. (Secret)

21. Arnold, R. G., et. al., "MMBC Emulation Design Requirements," Final Report, Vol. 6, Contract DASG 60-76-C-0049, March 1977.

22. System Development Corp. report TM-HU-242/005/00, 15 June 1977, p. 7.

23. Gouda, M. G., Han, Y. W., Jensen, E. D., Johnson, W. D., and Kain, R. Y., "Towards a Methodology for Distributed Computer System Design," Sixth Texas Conference on Computer Systems, November 1977.

APPENDIX A

DP DESIGN REQUIREMENTS STUDIES*

In this section several designer experiments are discussed that will address critical issues concerning data processing design techniques.

A.1 EMPIRICALLY-BASED EXPERIMENTAL DESIGN

The following aspects of experimental design, as concerned with BMD DDP design technology research, should be experimentally examined:

- Specification of experiment goals and requirements
- Selection of the test object (the object to be designed during the experiment)
- Specification and control of the test object requirements
- Control of experiment
- Selection and interrogation of experimenters
- Evaluation of resulting designs
- Evaluation of the design technology
- Characterization of test objects and experimenters for generalization of results for other applications
- Identification of experiment complexity limitations and supporting tool requirements

---

*This appendix was prepared by General Research Corporation.

PRECEDING PAGE BLANK-NOT FILMED

The studies should include design of associated questionnaires for collection of design experience data from experimenters.

A.2 ANALYTIC EXPERIMENTAL DESIGN

To support the "pilot" experiments above and future research aspects of experimental design theory requires analytic study. The aspects are expected to involve derivation of formalisms for defining and analyzing design parameters (or attributes), and derivation of "merit" function(s) to quantitatively evaluate designs (especially designs accommodating multiple requirements).

A phase of experimental study is now appropriate for verifying results and extending understandings of design requirements. In this section, we consequently define a series of design technology experiments which are planned to satisfy near term research requirements.

The first experiment will involve a single designer working on a single test object. At the conclusion of the design phase, the designer will be interrogated, as discussed above, as to the value of the design abstractions and the format in which they are organized. The result of this interrogation will govern the design of a questionnaire to be used in later multi-designer experiments.

If problems arise with the abstractions or format, modifications will be made before proceeding. If the modifications are major, the first experiment will be rerun with a different test object. If no or only minor modifications are necessary, an experiment will be conducted with multiple designers working on a single test object. For this experiment to be successful, care must be taken to present a common set of clear, complete, and consistent specifications to all designers (Volume IV of this document and Gouda, et al.).[23] Again, if the results of the experiment indicate that modifications in the abstractions are necessary, they will be made at this time.

The first two experiments will be evaluated using only qualitative measures. The measures will be primarily user-derived by means of the questionnaire, but might include some product-derived measures as well. Product-derived measures are most useful when there is a control group for comparison purposes. However, design consistency can be checked by examination of the product by an independent designer(s).

No control group was included in these first experiments since the primary purpose is to shake down the design theory formulation and the experimental procedures, especially the questionnaire and the identification of appropriate qualitative measures. It seems appropriate that a control group which does not use these design procedures should be included in the next phase of experimentation.

Work will proceed on the development of evaluative models and their associated quantitative measures concurrently with the first phase of experimentation, as well as on further refinement of the design theory. Both of these efforts can be expected to contribute to the design of the next set of experiments.

We plan experiments to contain the following steps:

1. Identification of Test Objects

   The first step in a proposed experiment is the identification of a set of prospective test objects for our design analysis. These will be chosen from the spectrum of BMD DDP tasks or BMD-like DDP tasks. Candidates for early consideration include the BMDATC Computer Independent Software Specifications (CISS), multistatic radar, Safeguard, and Site Defense. In studying these systems, attention will be given to the commonality found among them, keeping in mind the goal of identifying a set of generic tasks which describe essential portions of the BMD system requirements.

2. Identification of Measures and Evaluation Procedures

   In order to answer questions which help to evaluate design abstractions, it is necessary to specify measurement and evaluation procedures for the experiments. The set of possible measures can be divided into two classes: a) those derived from the user of the abstractions, the designer, and b) those derived from the design product itself, the results of the design.

a) User-Derived Measures--These measures tend to be comparative and subjective in nature ("A is better than B, but I prefer C to either."). The results may be quite variable from designer to designer and are probably affected by differences in designer qualifications. However, despite these weaknesses, these measures are reasonably easy to design and are probably more consistent than are simple objective evaluation models. An overly simple objective model may well give the wrong answer, whereas a subjective measure would be more likely to give inconclusive results than a wrong answer.

Such measures are obtained by means of interrogation and use of questionnaires. The questionnaires may include both structures (rank ordering of alternatives) and unstructured (free form comments) portions. The development of an appropriate questionnaire requires considerable experimental design effort to formulate and evaluate the set of questions. A probable progression is from free form interrogation to a prototype questionnaire which is tested on a subset of subjects and/or test objects followed by design of a final questionnaire.

b) Product-Derived Measures--Two kinds of measurements can be made of the design product: subjective and objective. The former involves evaluation of the design by a person(s) other than the designer and would result in rank ordering of designs and other qualitative judgments. Such an evaluation shares most of the advantages and disadvantages of the user derived measures.

Objective measures of the product require development of evaluation models which quantify performance with respect to the design requirements vectors. Given such measures of the "-ilities," it is still necessary to provide means of comparing admissible* designs to decide which is best. This implies that a total "cost function" needs to be developed which integrates the various requirements into a single measure. All of these steps will require significant effort. Hence, the first experiments will have to be evaluated subjectively, concurrently with the development of objective models.

---

*An admissible design is one which is not dominated by any other design, that is, compared to any other design, the admissible one is better in some areas and worse in others.

3. Application of Design Technology

As discussed above, the first step in the proposed plan is to select a design test object from the space of existing or proposed BMD tasks. Next, that system must be described in terms of functions, data abstractions, and predicates. This requires definition of the interrelationships and interface rules between the quantities and involves:

a. Construction of a Data Flow Graph, expressing the use and production of data by functions and the use of data to establish conditions.

b. Construction of a Precedence Graph, expressing the (partial) ordering between functions, in terms of their starting and finishing events only (if this detail is not sufficient, it implies that the functions should be refined into smaller units).

c. Description, in narrative form, of the intended use, restrictions, priorities, conflicts, objectives, goals, etc. of these quantities.

At this point, it is first possible to address the above goals for the design process, namely, feasibility, completeness, convenience, and utility. It is also appropriate to examine the problems caused by the lack of uniqueness of the specification process. What is the effect of defining a different set of functions, i.e., a different grouping of subfunctions? How and why does one do functional refinement?

The next step is to map this specification data into:

a. Abstract Processes

These basic units of parallelism in our specification format are composed of functions which execute serially at the current level of abstraction (but may have internal concurrency which can be exploited at implementation time).

b. Abstract Classes

These data abstractions, dedicated to a single process, can be used by any function of the host process. Performing this mapping requires specification of the operations upon the data, the legal ordering of operations, and the semantics of these operations, including the intent, access rights, etc.

c. Abstract Monitors

These data abstractions are shared by processes and can be used by any of the processes sharing it. They enforce mutually exclusive operations and require specification of the ordering with respect to components sharing it, the semantics of operation including intent, access rights, nature of scheduling, etc., and the scheduling of processes sharing it.

The form for specifying the interrelationships and interface rules for these data abstractions is not well established at this time, and will be the subject of further investigation as the experiment progresses.

4. Evaluation and Iteration

To apply the design technology the designer has to take a number of creative, or "engineering judgment," steps. For example, the designer must identify potential parallelism between functions and use that information to assign functions to processes. He/she must evaluate whether all specified functions are necessary as functions, or whether some are really data or functional control operations which are better implemented as monitors. He/she must also consider whether the level of functional refinement is appropriate and redivide the functions if it is not.

The designer must decide the questions of sharing vs. dedication for the data abstractions, and must decide how to combine or partition data abstractions so as to meet the design requirements. In particular, he/she must examine the specifications for growth and change to decide the appropriate degree of modularity and to answer the question of assignment to monitor or class abstractions.

The designer must also map precedence, priority, and performance requirements into scheduling rules, which prescribe the necessary control that will be embodied in the processes and monitors. He/she will look at the question of hardware/firmware/software implementation of control to see if the choice of algorithm or process organization will affect (i.e., simplify) the implementation.

An analysis of these creative steps provides evaluations of both the design product and design technology. Also, the design product analysis will often lead to design iterations, which are very important indicators of quality of the design technology.

The evaluations are to be performed using measures and procedures defined in Step 2, above.

5. Experimental-Design Iteration

The results of each step of each experiment will provide evaluative information about the planned experiments themselves. We expect to refine all of the experimental steps described above during the course of the research program.

A.3 DESIGN AND DEVELOPMENT TECHNOLOGY

The mutual impact of DDP design requirements and existing techniques is discussed in a companion report, DDP Design and Development Technologies Assessment. The overall results indicate that software (or process) engineering, performance validation, and SETS development and usage are much more complex for DDP because of the increased number of design variables; for SETS, the problems appear mainly at low levels of design (i.e., at implementation of the distribution).

Research and/or experimentation in DDP software (or process) engineering and performance validation is considered to be an integral part of DDP design technology studies, described above. SETS issues are more distinct

and relate to lower levels of design than those being considered in the DDP design technology studies. Consequently, SETS research and experiment plans are discussed further below.

A number of the critical SETS design issues can be at least partially resolved by experimentation, particularly in the area of timing control and inter-process communications. Some experiments are proposed below; they are by no means an exhaustive set, but rather are a starting point from which more elaborate experiments can be designed. The purpose of each experiment is presented, as well as the software requirements for its completion.

A.3.1 Stop/Start a Single Processor

An experiment that must be carried out to determine the feasibility of interrupted real-time testing requires that a single processor be stopped and then started a number of times in succession. Measurements must be taken so minimum, average and maximum shutdown and restart delays can be derived. The ideal mechanism for performing this experiment would be through use of hardware specially installed to interrupt the processor on receipt of an externally generated signal. In the absence of this capability, a software-implemented interrupt, which suspends processing and recognition of all interrupts other than a restart signal, should be used.

The process to be interrupted should resemble as nearly as possible a BMD real-time software module insofar as its input/output activity is concerned. The I/O handlers must be permitted to continue operation if the real-time interrupt occurs during a data transfer.

A.3.2 Stop/Start Multiple Communicating Processes

Following the successful starting and stopping of a single processor, the same operation should be attempted on two and then additional communicating processors. First, the process should carry out a function to completion without interruption to set down baseline results for comparison. The next step is to successively stop and restart the processors until the job completes.

One purpose of this experiment is to determine how far each processor's idea of current engagement time drifts from the true in the absence of a periodic synchronization pulse. Other results of interest include the degree of difference in I/O activity between the continuous and interrupted runs, whether any messages were lost or seriously delayed, and how different the message sequences were.

A.3.3 Repeatability in Restarting

One or more experiments should be devised to determine how closely matched orders stream from successive runs of a multiple-processor, simulated real-time processors. This experiment must show that successive runs of the RTP produce similar orders streams if repeatable real-time is to be a feasible testing strategy. To be meaningful, more than one processor should be used (even though initial runs using a single processor should be made); each processor should produce its own SETS-related orders sequence.

The experiment should eventually include runs with a SETS module in closed loop operation so that later orders in the data stream will reflect data appearing in earlier returns. To accomplish this, a BMD algorithm, such as radar scheduling or target tracking, should be simulated in the RTP. The degree of success of this series of experiments will be measured by the closeness with which successive streams of orders match.

A.3.4 SETS Operation for Restarting

It must be shown that a centralized SETS system can retrieve and transmit prestored returns to the RTP significantly faster than they could otherwise be computed before RRT is deemed feasible. This experiment assumes that repeatability of the RTP has been proven successful (see the experiment in Subsection A.3.3). It remains then to determine how many additional orders can be processed by SETS beyond those with prestored returns.

In addition to the RTP required in the experiment in Subsection A.3.3, this experiment requires a SETS program with the following capability:

1. A returns preparation algorithm representative of a distributed system of interest.

2. Logic to retrieve and transmit prestored returns to the RTP when it is determined that they correspond to recently received orders.

3. A policy on preparing returns for unexpected orders.

4. A method for determining if the program is approaching an impending returns deadline, in which case it can stop the test, finish its calculations, and initiate a restart.

This experiment will also produce data from which an extrapolation can be made to determine the maximum number of distributed processes that SETS can support in an open loop testing configuration. This is possible because RRT is simply a method of converting closed loop into open loop testing.

A.4 SUMMARY AND SCHEDULE

Experimental efforts during FY 78 should be directed towards resolving the issues described in Subsection A.2.1 by performing the experiments outlined in Subsection A.2.2. Figure A-1 is a proposed schedule of these activities.

Section A.2.3 outlines results of studies in design and development technologies, and describes some basic SETS experiments. We propose that the basic SETS experiments be performed in FY 79, along with detailed studies of design tool and language requirements (see DDP Design Technology Research Requirements Definition, Section 5, for further discussion). Schedules and plans for these efforts should be further detailed during FY 78.

PROPOSED TASKS

1. FIRST EXPERIMENT
   - SELECT TEST OBJECT
   - DEFINE MEASURES
   - APPLY TECHNOLOGY
   - EVALUATE/ITERATE
   - INTERROGATE DESIGNER
2. SECOND EXPERIMENT
   - SELECT TEST OBJECT
   - SELECT EXPERIMENTERS
   - APPLY TECHNOLOGY
   - EVALUATE/ITERATE
   - INTERROGATE DESIGNER(S)
3. ANALYTIC EXPERIMENT DESIGN
   - MEASURES/CRITERIA
   - EXPERIMENT CONTROL
   - MERIT FUNCTIONS
   - QUESTIONNAIRES
4. INTERFACES DEFINED

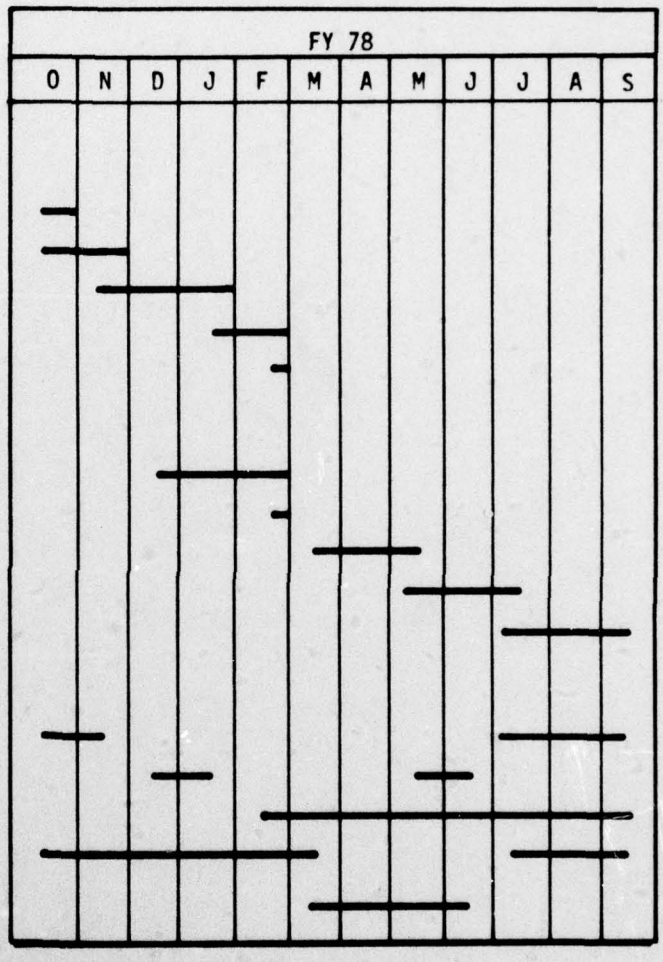


Figure A-1. Proposed FY 78 DDP Technology Experiment Schedule