Allan, John J

THE UNIVERSITY OF MICHIGAN

INDUSTRY PROGRAM OF THE COLLEGE OF ENGINEERING

MAN-COMPUTER SYNERGISM FOR DECISION MAKING IN THE SYSTEM DESIGN PROCESS

John J. Allen III

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in
The University of Michigan

August, 1968

IP-821

PREFACE

The subject of this dissertation is the engineering design process. The purpose of the investigation was to delineate some basic implementation principles for a computer-based interactive design system which would enable the designer to be more effective in his work.

The thesis of the work is: design tasks must be done within time and economic constraints. A designer makes his decisions based on his best available information which is, in part, a function of how well he can communicate with his environment. His information interchange with his environment can be facilitated with the proper computer interface, thus allowing the designer an opportunity to make a greater number of effective decisions per unit time. More decisions per unit time means that some otherwise ignored or grossly estimated component interactions of the system which the designer is designing may now be evaluated more quantitatively.

To the extent that the design process has been identified, a design system to augment the effort of the engineering designer is postulated. An implementation of the postulated design system has been built and used. The conceptual method of operation is described in Chapter 5. The implementation is discussed in the Appendices.

This research itself was approached using the methodology of Chapter 2. The original needs statement was: "What does it take to do a better job of designing a complex, real-world system?" The key elements of an answer to that question

seem to be:

1) a fast graphical way of manipulating design problem topologies,
2) an implementation of the Vari-port link concept,
3) an associative central computer data structure for speed, and
4) a file-oriented central computer system to provide adaptability.

Some "solved engineering problem results" are discussed in Chapter 6, along with several aspects of the relevance of the work to practicing engineers. Conclusions and some open problems are presented in Chapter 7.

This research was carried out with the cooperation and assistance of many people. The author is most grateful for the advice, continuous interest and counsel received from the members of his doctoral committee. The author is especially indebted to his Chairman, Professor Frank Westervelt, for his friendly encouragement, helpful criticism and most importantly for providing constantly stimulating intellectual challenges in areas related to the work. The reviews of the work by Professor Bernard A. Galler and Dr. Marvin T. Ling, Graphtek Corporation, are also sincerely appreciated.

The author will always be indebted to Dr. Gordon J. Van Wylen who as mentor has also been a constant source of encouragement since the start of the author's graduate work.

Mr. James H. Jackson of the Systems Engineering Laboratory, the University of Michigan, has been an invaluable friend.

In addition to giving the author the algorithms for the SEL/338 executive of which Mr. Jackson is originator, he spent many hours of his own time to help insure that the system was operational.

Mr. Jack R. Guskin has been a very good source of ideas relative to the central computer implementation. The author is very grateful for his enthusiasm and devoted assistance.

This work would have been much more difficult without the continuing assistance of Miss Kristina Hermann, who in all matters of computer file updating and typing was able to assist while keeping the material in context. The typing and outstanding draftsmanship of Mrs. Heide Malhotra is sincerely appreciated, as are the many helpful suggestions from other Concomp Project staff people.

The author is very grateful for the financial assistance provided during the period of this research by The American Oil Foundation, The Ford Foundation, The University of Michigan and the Advanced Research Projects Agency of the Department of Defense, through the Concomp Project (contract number DA-49-083 OSA-3050, ARPA order number 716) administered by the Office of Research Administration, The University of Michigan.

Finally, kudos are due the author's wife Ferne, who continued her career as an outstanding research engineer while doing much more than her share to keep our family on an even keel during the sometimes trying time periods experienced.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (CONT'D)

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

"...engineering design is a particular kind of problem solving—science-based problem solving with social-human awareness." "This application of knowledge is every bit as respectable an intellectual challenge as is the acquisition of knowledge. The application of knowledge requires understanding and purpose."[1]

John R. Dixon

Engineers are continually required to solve problems. The solutions to these problems are usually new ways of adapting our surroundings to our needs. If these problem solutions come to fruition as physical contrivances, one tends to say that the engineer must have done some design work. In recent years it has even become acceptable to use the word "design" in connection with esoteric solutions.

The engineer who tackles problems so broad and complex that most specifications of need are not initially obvious has come to be called a systems engineer. He brings order to problem definitions, and devises operations to match the functions that add up to solutions. He sets criteria on the functions so that intelligent tradeoffs can be made, and operates from a point of view that recognizes functional similarities.

In Chapter 2 the information interchange between the designer and his environment is interpreted. One of the designer's major efforts seems to be the synthesis of alter-

---

[1] John R. Dixon, Design Engineering: Inventiveness, Analysis and Decision Making (New York, McGraw-Hill, 1966), p. v.

native methods of implementing an idea. This leads to trial solutions in attempts to make the system outputs meet the overall specifications of the problem. Another major effort seems to be the analysis of these alternatives to fortify his decisions.

A wide variety of analysis techniques is available to help the designer examine the synthesized solutions. Hence students of design should devise tools
that will allow other designers to consider the many inter-component interactions that result from the multi-component nature of many contemporary design problems. Engineers should have a system with which they can interact in the design situation so that more of these feedback aspects of complex problems can be quantitatively considered. The need for this inclusion of more component interaction in the design process is the theme of systems engineers.

Consider that design tasks must be done within time and economic constraints. A designer makes his decisions based on his best available information which is, in part, a function of how well he can communicate with his environment.

Prior to beginning this work, it was assumed that the designer's information interchange with his environment could be facilitated with the proper computer interface. This would allow the designer an opportunity to make a greater number of effective decisions per unit time.

1.1 Background

In the past twenty years or so, two of the more important concepts which have

emerged, are information theory and system theory. Each of these concepts is extremely important in its own right and has many different applications. The theme of this work is that they should be considered together. By a joint study, each concept can gain from the other; each concept can make large contributions to the other.[2] "For example, analysis of the information required for the various phases in the life of a new system reveals some tasks that computers might perform, some that human beings will always have to perform, and some that either may perform depending on the circumstances."[2] "...in theory, computers can perform any specified task that does not contain physical impossibilities or logical contradictions."[3] According to the definition of the word "machine," computers are machines; hence, computers should be able to perform some of the tasks of system design.

1.1.1 Systems and Networks

Mechanical networks are members of a far more general class of objects called physical systems. Physical systems exist in the real world and have the common characteristic that certain forces, called inputs, are transformed in real time by the system into responses called outputs.

Men such as Leonardo da Vinci, when designing his woodworking machines, performed many of the functions of today's

---

[2] Ira G. Wilson and Marthann E. Wilson, Information Computers and System Design (New York, Wiley, 1965), p. ix.

[3] Ibid., p. 10.

system designers. It is true that some contemporary systems are more complex, and possibly even spread over far larger areas; however, the fundamental design steps are much the same today as they were in earlier centuries.

While systems have been used for centuries, the "systems concept" is only about twenty years old.[4] This concept considers all the components of one system instead of individually considering separate subsystems. "The system concept is useful because it integrates the separate requirements so that the resulting overall ... system ... (can be) optimized."[4]

A system is a set of operations organized to satisfy definable user requirements. An "operation," here, means performing a practical work involving the application of principles or processes. Therefore, this definition of a system includes not only structures of tangible black boxes, but manual procedures and computer programs—intangible but very real structures. The definition also implies satisfying human users and a way of measuring the effectiveness of a system. The value of the "procedure" in system design is that it shows that solutions can be made to depend on successful information retrieval, logical operations, and computations, rather than on inspiration.[4]

Why Systems Came to Be

In order to deal with physical systems in any mathematically meaningful way they must be abstracted. The usual

---

[4] Ibid., p. 182, 198

process of abstraction is to decompose the system into an interconnection of idealized components, each of which is precisely defined mathematically.

Most engineers are aware that the physical phenomena they study are composed of sheets of metal, pieces of glass, etc., with electric, thermal and other fields filling the whole of space. But such systems can usually not be described continuously when communicating to other people the values of the fields' strengths and other properties at every point in space. Potentials, fluxes and other magnitudes can be measured only at a finite number of places. These "places" form the set of data for discussion. Hence, "...quantization into a finite set of attributes is a logical necessity of description."[5]

Lumped mechanical and electrical networks form such quantized abstractions; they possess a finite number of elements and a finite set of relationships between them. When expressed mathematically, the equations have a finite set of variables and coefficients. This process is called "modeling," and is a necessary part of any real engineering systems problem. The abstract system may then be defined in more mathematical terms, as, for instance, a set of ordered pairs of vector-valued functions of inputs, outputs and time. When a system is described in this manner, we say it is characterized by an input-output relationship. In essence, then, network or system theory is the study of how interconnected

---

5 Proceedings of the Symposium on Information Networks (Brooklyn, Polytechnic Institute of Brooklyn Press, 1954), p. 177.

components perform as a group for various topological arrangements which we will call topologies.

In the early 19th century, Kirchoff recognized the uses of topological and matrix methods for the calculation of the properties of electrical, mechanical and fluid networks. The mathematics associated with this calculation relates to graph theory. The first paper on this division of mathematical topology had appeared in 1736 by Euler. First attempts to develop Kirchoff's suggested methods were slanted toward electrical network analysis and date back to the 1920's and the works of Kron. However, prior to Kron, Maxwell developed his field concepts in the mid-19th century. Poynting extended Maxwell's recognition of the energetic aspect of fields to the consideration of energy transport in an electrical network. On the basis of Poynting's work, Steinmetz laid the foundation of electrical engineering circuit analysis and design. By the beginning of this century the analysis of the macroscopic behavior of multi-component problems on an energetic basis had become quite acceptable. However, the advent of quantum and relativistic mechanics diverted attention from the theory until the 1940's when interest renewed.[6] These methods of energetic system analysis received continued attention in the 1950's and early 1960's giving rise to the works of Darlington,

---

[6] The rationale of the above discussion was derived from the following works: Henry M. Paynter, Analysis and Design of Engineering Systems (Cambridge, Mass., M.I.T. Press, 1960); Oystein Ore, Graphs and Their Uses (New York, Random House, 1963); W.H. Kim and R.T. Chien, Topological Analysis and Synthesis of Communication Networks (New York, Columbia University Press, 1962).

Huelsman, Kim and Chien, Seshu and Reed, and Wineberg, which are oriented toward electrical networks.[7]

Although Kirchoff had looked at general networks, most of the applied work up to the mid-1940's was in the electrical network area. One can postulate that the problems created by World War II, from which operations research evolved, were also of concern to the scientists and engineers dealing with non-electrical multi-component problems. It would have been natural to take a well-developed method for handling a specific kind of network and want to apply it to networks in general. Schöenfeld[8] published the development of the universal analogy between hydraulic, mechanical, acoustic and electrical systems in 1954.

Since the mid-1950's the network approach, or more explicitly the notion of considering the component interconnections of a multi-component problem, has been given much attention in non-electrical circles. Works representative of these efforts that are specifically oriented to discrete component problems are those of Fenves and Branin, Lind, Paynter,

---

[7] W. Darlington, "A Survey of Network Realization Techniques," I.R.E. Transaction, CT-2 (February 1965); R.G. Huelsman, Circuits, Matrices and Linear Vector Spaces (New York, McGraw-Hill, 1963); W.H. Kim and R.T. Chien, Topological Analysis and Synthesis of Communication Networks; S. Seshu and M.B. Reed, Linear Graphs and Electrical Networks (Reading, Pa., Addison-Wesley, 1961); L. Wineberg, Network Analysis and Synthesis (New York, McGraw-Hill, 1962).

[8] J.C. Schöenfeld, "Analog of Hydraulic, Mechanical, Acoustic and Electrical Systems," Applied Scientific Research, Sec. B, Vol. 3, No. 6, (1954) pp. 417-450.

Pullen, and Shearer, Murphy and Richardson.[9]

Finally, works began to appear several years ago by a group of authors who view all engineering problems as important and complex enough to have relevant component interconnections. These authors are the systems engineering writers whose ideas are directed to the worthwhile and necessary goal of more inclusive analysis as the key element of better engineering design. Their idea is that the designer can make better decisions if he has better information on which to base them. Representative of the more concrete of these philosophically-oriented works are those of Eder, Hall and W. Wilson.[10]

System Constituents

System analysis encompasses the integration of the study of the effect on systems of both material and information inputs and outputs. "In mathematics, the term "canonical forms" refers to the simplest and most significant forms to which general equations may be brought without loss of generality..." "...using this meaning of the word canonical, it is ... possible to represent a canonical model of a complex system. If

---

9 S.J. Fenves and F.H. Branin Jr., "A Network Topological Formulation of Structural Analysis," (IBM Technical Report No. 00.979-1, September 1964); N.C. Lind, "Analysis of Structures by Systems Theory," Journal of the Structural Division, A.S.C.E., 88 (April 1962), p. 1-22; H. Paynter, Analysis and Design of Engineering Systems; K.A. Pullen, Theory and Application of Topological and Matrix Methods (New York, Rider, 1960); J.L. Shearer, A.T. Murphy, H.H. Richardson, Dynamic Systems, Pre-Publication Edition (Reading, Pa., Addison-Wesley, 1965).

10 W.E. Eder, Mechanical System Design (New York, Pergamon, 1965); A.D. Hall, A Methodology for Systems Engineering (Princeton, N.J., Van Nostrand, 1962); W. Wilson, Concepts of Engineering System Design (New York, McGraw-Hill, 1965).

viewed broadly enough, all complex systems have both inputs and outputs of zero entropy energy, information carrying energy, and material objects. Hence, it follows that system differences more or less depend on the interests of the system designer."[11]

In the next section, the ideas of information and entropy are discussed in greater detail; however, several points relevant to the present discussion should be made at this point. When energy flows from a source to a receiver, if no new knowledge is gained by the receiver, then no information has been transmitted. Further, the word entropy has a probabilistic connotation with respect to information just as it does with respect to thermal systems. It refers to "...the average language information per symbol or per message as the case may be."[12]

"All systems must have an input of information-carrying energy—even if it is only to start the operation. All systems must (also) have an input of zero entropy energy because by definition, information-carrying energy is modulated zero entropy energy. All systems must have an information output which may be either in the form of information carrying energy or of observable changes ..."[13] in material objects. All systems must have a zero entropy energy output too, because

---

[11] Ira G. Wilson and Marthann E. Wilson, Information Computers and System Design, p. 114.

[12] Stanford Goldman, Information Theory (New York, Prentice-Hall, 1953), p. 43.

[13] Ira G. Wilson and Marthann E. Wilson, Information Computers and System Design, p. 114.

all systems have unwanted losses. However, because unwanted losses can be observed, they may convey information to another system. When examined closely, all systems have an input and output of material,because in order to create zero entropy energy, some fuel must be used.[13] Consequently, system analysis can be said to encompass the integration of the study of the effect on a system of both the material and information inputs and outputs.

1.1.2 Information Theory

Today, information theory is still basically communication theory whose fundamental "...problem is to reproduce at some point or points, either exactly or approximately, a message selected at another point. The important word is selected, because the actual message is assumed to be selected from a set of possible messages. Stated another way, the transmitted message must be unexpected and hence unpredictable because if the receiver knew what it was beforehand, no information whatever would be gained by receiving it. Thus, the amount of information conveyed by a message depends on the unexpectedness of that information."[14]

In 1948 Claud E. Shannon, research mathematician for the Bell Telephone Laboratories, published a paper entitled "A Mathematical Theory of Communication" and thus became the father of the new discipline. In the same year, Norbert Wiener published Cybernetics. Both works arose to

---

[14] Ibid., p. 254.

some extent from the authors' occupations during World War II.

Existing information theory is based on mathematics. Definitions are precise and theorems are proved rigorously. However, system theory is in quite a different state. Mathematical computations are used in some cases, but today, system design and development are really more an art than a science. If studying system design in terms of non-rigorous information theory can be allowed, some answers might be suggested to the problems arising in designer-environment communication.

Whitehill points out that, "One test of the plausibility of borrowing from one discipline to enlighten another is to investigate the extent to which the small terms—the building blocks—of the one may, without forcing, be taken as analogs to similar terms of the other. For example, the language, dimensions and relations of perfect fluid hydraulics were early found analogous to the partitive terms in electricity."[15] Students who first learn the principles of hydraulics find the principles of electricity easier to learn. "The usefulness of that analogy is not diminished because it breaks down when higher frequency alternating currents are studied: it has already served its purpose."[15]

The following lines are paraphrased from Claud Shannon's article on information theory, in the 1967 edition of the Encyclopaedia Britannica:

---

15 Joseph Whitehill, "Reappraisals: II - Samuel Taylor Coleridge: Prisoner and Prophet of System," The American Scholar, 37, 1, (Winter 67-68) p. 151.

> A study has been made of the distribution of lengths and frequency of occurrence of different words in a language such as English. The analysis results of this experimental data can be explained as a consequence of the assumption that a language gradually evolves under continued use into an efficient communication code. These results suggest that the human being may tend to adopt something like the ideal encoding occurring in communication theory.

These results further suggest using simple one-to-one analogies between the equipment and processes of communication theory, and the graphic devices and means of engineering language. That is, good technical prose is more effective communication than bad technical prose, and drawings are more effective than either; and information theory might offer a way not only to tell the good from the bad, but also to tell why.

Of all the designer's tools, the one of singular pre-eminence is graphics, for through graphics he can communicate more effectively, and only through communication does he progress. The management of language and graphics is part of the science of communication. The study of the management of design language and graphics is an effort to develop an information structure of design.

It would seem odd if mathematics had nothing to contribute to the art of engineering design. The idea is so obvious that many have tried to evoke the entire field of design from mathematics. This seems to be a doubtful endeavor in that it tends to look at material entities rather than at the human beings who work with and appreciate them.

1.2 Considerations of the Design of a Design System

When considering the problems of the design of a design system, the concern arises that one may not only devise a system focused outside man on the end product, but one may also so rigidly structure the enquiry, the analysis, and the decisions, that all hope of freedom of decision is gone for the ultimate user or designer who it is hoped will be served.

What is really sought is a system with two characteristics: (1) the system itself will not structure either the user's view of the environment of his problem, the formulation of the problem, or the solution of the problem; (2) the design and decision processes will be designer-oriented, and an integral part of each problem itself. In the construction of a computer-based system for aiding the designer, it is important that the constraints that the system imposes on him are not so severe that the man who originally programmed the computer actually dictates the designer's problem solution. Only the user/designer can perform the function described as "invent-discover-develop."

Esherick points out that the physical sciences, with their isolatable phenomena are, no matter how complex, far simpler than the problems in the social sciences; they are less ambiguous, more testable, and the entire process can be communicated with an entirely different kind and degree of precision. "The very ambiguity of the social science problem, and design is a social science problem, is its woolliness, its vagueness, and its unwillingness to be pinned down. All

of these elements of the problem are, in fact, elements in life itself, and whatever is done in the design of a design system must preserve this capacity for ambiguity."[16]

Thus, while mechanical systems adapted from the physical sciences may be useful for certain well-defined and restricted parts of any design system, their very nature prevents their use for the entire design system. Certain human acts are indispensable. A human must conceive the need for the new system. He must decide whether to start doing something about it. Estimating the odds for success, the costs, and the times required for complex design tasks is to some extent an art. A human must decide on or choose the system requirements, and he must put the system requirements in preferential order. This implies not only that the design of a design system should be generated from within the domain of design, but that the system users must be incorporated in the design system plan.

In further commenting on the system design process, Esherick feels that there are two general purposes for design. The first is when "the entity to be designed is to be used directly or overtly for known or knowable processes."[17] The second is when the entity "is designed for complex and contradictory functions where the entire future growth process is

---

16 Joseph Esherick, "Problems of the Design of a Design System," Conference on Design Methods (New York, Pergamon, 1963) p. 76.

17 Ibid., p. 78.

unknown and unknowable, ... where the primary objective is to minimize control, to generate maximum freedom of communication, freedom of motion, freedom of choice, and where the maximum conflict and contradiction will be possible ..."[17] Consequently, a system to aid the designer must be conceived so that it is open and flexible, itself possessed of the ability to grow—adaptive. This can be accomplished, it seems, if the design goals of the system are themselves open.

1.2.1 The Designer's Quest for Aid

For centuries designers sought better ways to express their ideas and better ways to perform analyses to fortify their decisions. Early drawings were of the "layout" type. In fact, they were carvings not drawings. As units of measure were agreed upon, the physical dimensions of drawings took on a new significance—they became data.

As the fundamental principles of element design were developed in more recent times, another whole new area of concern began to evolve. If calculations could be made to substantiate the validity of a drawing's dimensions, one would then have to know the physical properties of the entities. Hence, the designer began a search for better ways of handling his reference data—today known as information storage and retrieval.

The analyses referred to above have become more sophisticated over the centuries as mathematics has been developed to allow a closer description of the physical world. Accompanying this trend has been an effort to mechanize the

solution of the ensuing mathematical expressions. Much time elapsed between the inception of the abacus and the early mechanical computer. However, relatively little time has passed since the electronic digital computer became a reality and progressed to the state of multi-user multi-programming capability.

In all of the above discussion, the underlying drive has been man trying to adapt his surroundings to his needs. It is also interesting to note that his problem categories are still the same today—analysis, data storage and retrieval, and the expression of ideas.

1.2.2 The Type of Problems Being Considered

In general systems parlance, a system is a group of elements which bear some relation one to another. In the work considered here, a system is several real components which are coupled energetically. Energetic coupling implies, of course, both material coupling and information coupling. The design of these types of systems is not styling, but involves parametric selection, iterative analysis and also form design as it affects analysis. The analysis involved is relevant to things that can be built, which in turn are composed of things that are or have been built. The former statement can be carried to very simple parts which probably have no analysis connected with them at all.

Things made of elements are called subsystems. Things made of subsystems are called systems. A system for one designer may be a subsystem for another. Therefore, a system

can be made of simple parts with no individual analyses, although the system *in toto* might require some analysis. Hence, the type of systems under consideration here is one made of one or more discrete components each of which may have one or more analysis considerations connected with it.

Because interconnected analyses are under consideration, and further, given that analysis is information processing, it is interesting to see what this view of problems could mean to a design system's information flow. Quite obviously the independent analyses must be able to communicate with each other in addition to being accessible to the user. Herein lies the challenge. The analyses' ways of divulging the same variable (formatting) are undoubtedly not matched. The physical units might need conversion. Further, one analysis may produce more information than the next cares to take in, or vice versa: a previous analysis may not have provided enough information. These are some of the major concerns that will be discussed in depth in Chapter 4.

1.2.3 The Idea of Ease of Man-Computer Interaction

Decision making is governed largely by economics in the design process. If a designer elects to employ a computer, the constraint of having to reduce input and output to character strings makes the use of computer facilities very difficult, if not impossible, for some problems. Communication in pictorial terms, however, can many times be rapid. Since inputs and outputs can conveniently be specified in two dimensions, or even higher dimensions projected onto a two-dimen-

sional surface, new problem areas can be attacked. Thus, through the use of display consoles, new freedom is provided.

In addition to providing new convenience and freedom, and better approachability, computer communication through display consoles makes possible an entirely new method of problem-solving. Under this arrangement, with the display surface serving as a common working area, the man and computer can work in prolonged, intimate, interactive contact. The display surface serves as a broad-band communication channel through which the user can rapidly ask questions and the computer can rapidly provide answers. The rapid answers presented directly to the man by the computer can stimulate the man's thoughts more than if the functions of the man and the computer were separated in time, or by intermediate personnel. The apparent reason for this is that with an unbroken chain of thought, a designer seems to be able to expend his energies on developing creative associations for the problem at hand, as opposed to re-initializing himself to his current state of achievement every time he is interrupted. Thus, a partnership of the man and the computer now becomes possible. Within this partnership, since significant time and economic penalties do not result from making a few wrong sorties, or from exploring paths that do not produce immediate results, a new freedom in problem solving arises.

In the previous section, the type of design problems being considered was outlined. In Chapter 3 the principles of computer augmentation of decision making are discussed in

detail. It is pointed out that the computer must play three roles if it is to augment the efforts of the engineering designer. Those are the roles of: (1) a teaching machine, (2) an analyzer, and (3) an information storage and retrieval system.

Before discussing these three roles, it should be noted that in each case there are two distinct connotations to the phrase describing that role of the computer. To the author's knowledge, these important and definitely distinct connotations have not previously been clearly identified or possibly even recognized.

As a Teaching Machine

First, engineering designers are generally not well versed in the intricacies of computer programming. It is difficult to design a system which does not require some knowledge of the digital computer or its input/output devices—as a trivial case: even "signing on" is work for a novice. Hence, in order to reduce the time which it takes an uninitiated user to become familiar with the system or to recover from an error which he has made, the system must coach and prompt the user as the needs arise. Therefore, one "teaching machine" consideration of a computer-based design system has to do with the operation of the system itself.

A second and entirely separate consideration has to do with the way in which the subject matter is accepted from and presented to the user. There is a fine line between the teaching machine characteristics under this title and some

of the characteristics generally attributed to an information retrieval system. In this case the message interchange is automatically context-dependent. The designer is actually helped with his engineering fundamentals.

As an Analyzer

The computer's role as an "analyzer" is sometimes the only role envisioned, and surely even for those who recognize the other roles outlined above, this role usually means mathematical analysis. To be sure, the computer deals only with symbols; however, the analysis or "interpretation" of "high level" (see section 3.2.1) input-output is a second and separate kind of "analysis" task. In this case the analysis is for the purpose of supplementing input or output information to make it readily communicable to the user.

As an Information Storage and Retrieval System

Here, the first and most commonly conceived mission of the computer is the deposition and recollection of stored facts. When using the information retrieval capabilities of a design system, a user knows *a priori* that he is going to exercise discrimination on the data that he is trying to retrieve. Consequently, he needs a system which will allow him to specify attributes of the desired information and selectively retrieve sets of it. Implementing this capability is discussed in Chapter 4, where it is also pointed out that the design system should accumulate information with use.

The second notion of "information retrieval" relates to the collection not of stored facts, but of parameters being

"picked up" (transducer monitored) from physical models for the purpose of having "real" parameters in the mathematical model. In this case, the computer helps "observe" the real models which designers use. This is not what is generally called "process control."

TDS (This Design System) Is These Three Ideas Combined

What the computer has done in the past to ease the burden of engineering calculations "will be done in the future for the whole area of engineering communications, decision-making and systems design."[18] This statement assumes that highly responsive automated systems based on powerful and large-scale information networks and totally new kinds of programming for manipulating the data will be available. The experimental work described in Chapters 5 and 6 is an effort in this direction.

TDS has been conceived as a blend of the three functions outlined in the above sections. Although in the experimental work reported no real-time data are monitored, provision for this capability is allowed. The basic criteria as each of these aspects of the system was designed and developed was that of tight coupling between man and computer. In this way, the man and machine taken together increase each other's effectiveness, the thought conveyed by the word synergism.

---

[18] C.L. Miller, "Some Comments on the Future Practice of Engineering," Proceedings of the Conference on Impact of Computers on Education in Engineering Design, Commission on Engineering Education.

CHAPTER 2

THE INFORMATION STRUCTURE OF ENGINEERING DESIGN

In Chapter 1 the point was made that flows in and out of engineering systems consist of both materials and information. Of course, nothing precludes the existence of a system whose flows of interest consist predominantly of materials or of information. Historically, mechanical engineers, for example, have looked at material flows. There generally are two classes of flows in their considerations: (1) those of a thermal nature, and (2) those of a "mechanical" nature. Only in the area of automatic controls do these engineers begin to intentionally process information. However, this is quite logical because the notion of handling information quantitatively has only existed since the late 1940's. Furthermore, information processing hardware for these engineers to "build into" their designs has only recently become readily available, and physically and economically feasible.

The computer-based system described in Chapter 5 and Appendix 3 is, quite obviously, an "information handling" system. In fact, as will be discussed later, the structural organization of TDS has been developed from the following customarily conceived concept of the design process, and this very development has been according to this same concept of the design process.

2.1 Information in the Design Context

Design is an information handling process. Consequently, as a process, it must have a structure. Hence, if

one is to build a system to augment the effectiveness of the designer, the information structure of design must be known. This structure is not the same as the customarily conceived sequence of activities used to describe the design process.

In the balance of this chapter, by looking at the design process, the elements of information that are brought to bear on a design problem are first delimited. The structure of that information is then outlined, and the notion of "processing" it is discussed.

As will be pointed out below, the design task is one of decision making. After discussing computer augmentation of decision making for design in the next chapter, general principles for developing the structural organization of a computer-based design system will be developed in Chapter 4 by melding, in perspective, Chapters 2 and 3.

2.1.1 The Designer as an Information Processor

Recalling that an engineering system is a group of components that bear some relation one to another; there are people called engineering designers who conceive of and develop systems of varying degrees of complexity. Given that a designer can be considered as an entity in an environment, then as in Figure 2.1, the designer may be depicted as a control volume. Several paths for information exchange with his environment are shown. Quite obviously, the graphic representation of the model could have been presented differently and still depict the information processing situation for design. However, represented here, and shown more explicitly in Figure 2.3, are the fewest

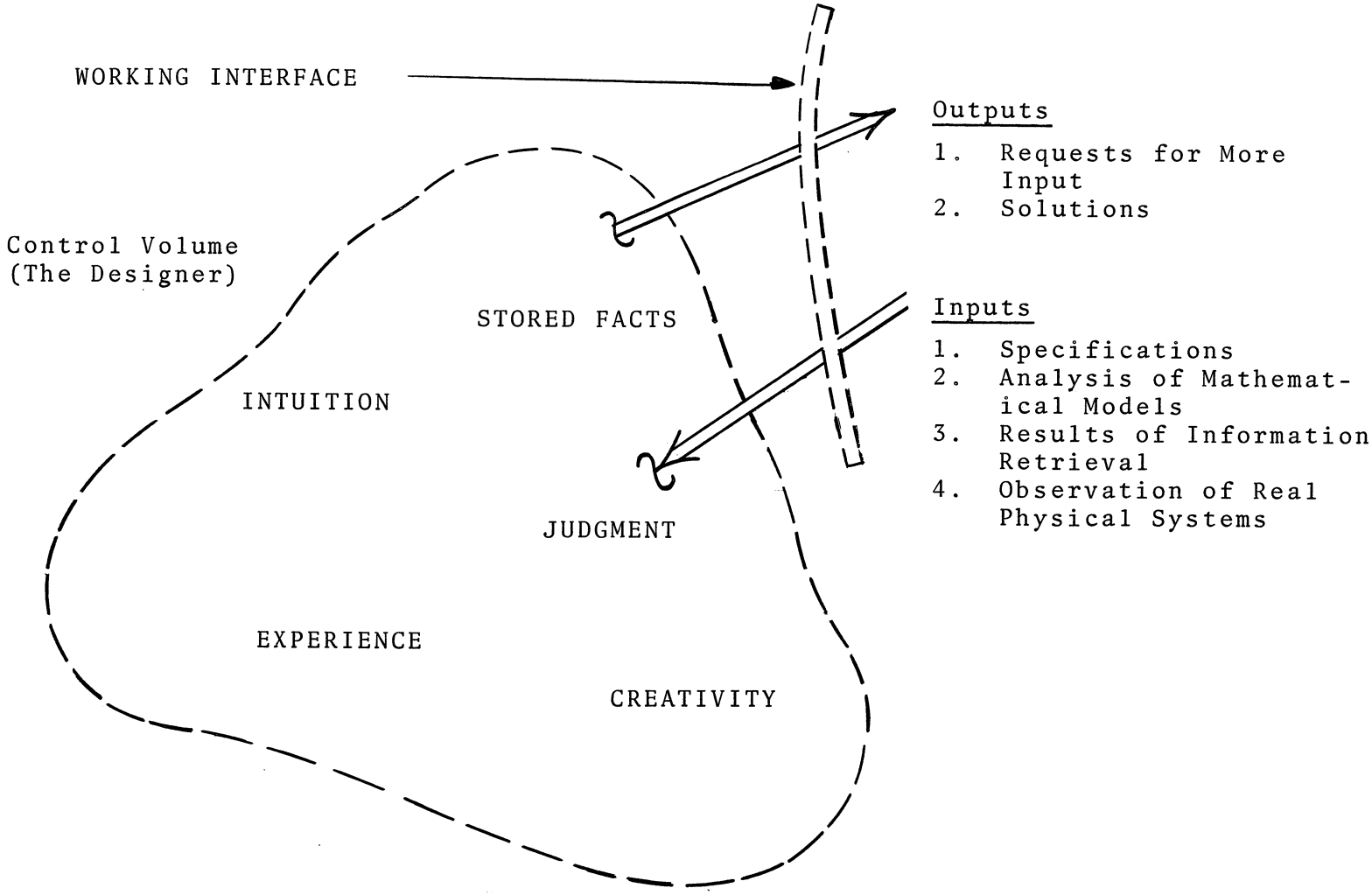


Figure 2.1. The Designer as an Information Processor.

number of meaningful design information flow categories that make this model usable for study. That this is true is demonstrated by an existence proof. Namely, this model has been used as the basis for developing the principles on which the design of TDS itself is based. The model is sufficient as TDS does indeed fulfill its basic objectives. That these flow-categories are necessary becomes obvious at the end of Section 2.1.

Now, with a designer viewed as a control volume, the flow through any control surface (in and out of any control volume) consists of information. Subsequent input information will convey a current problem of the designer's environment, and he will process information from several sources in order to generate output information which conveys a solution to the environment. These sources are: environmental inputs, intuition, creativity, experience, judgment, and stored facts about the physical world.

Exactly what goes on in the control volume is unknown. Furthermore, most systems' problems are difficult enough that the designer's first output is <u>not</u> the solution. Usually an information interchange with the environment takes place, i.e., the "input" source is activated by requests from the designer. The input that he then receives is either qualitative or quantitative, and consists of specifications, physical facts, or the results of tests.

These test results are the observed actions of activated models. These models are either entities in the physical

world (to some scale), or they are mathematical approximations. Models which are mathematical approximations require information processing by the designer in order to realize the input. This is known as analysis.

This quantitative information processing (analysis) governed for the designer by physical laws, often takes him a long time, or is impossible, "in the control volume." Relieving him of much of the analysis task, and allowing him to convey his ideas more readily, would leave him free to allocate the majority of his information processing abilities to other aspects of the problem under consideration. Because engineering problems must be solved in a finite time, and considering that information processing takes time, less time spent on analysis would mean more time for other things. These "other things" would be the processing of information previously neglected, or the values of which were assumed. Being able to take more system component interactions into account quantitatively would improve the effectiveness of the designer.

To this point, nothing has been said about a structure for the information that is brought to bear on a design problem. However, looking at Figure 2.1 once again, this information interchange manifests itself at the working interface as a five-step process. These steps are delineated in Figure 2.2.

2.1.2 Interpreting Flow Through the "Working Interface"

It is very desirable at this point to interpret the delimited information flow categories in terms of the custom-

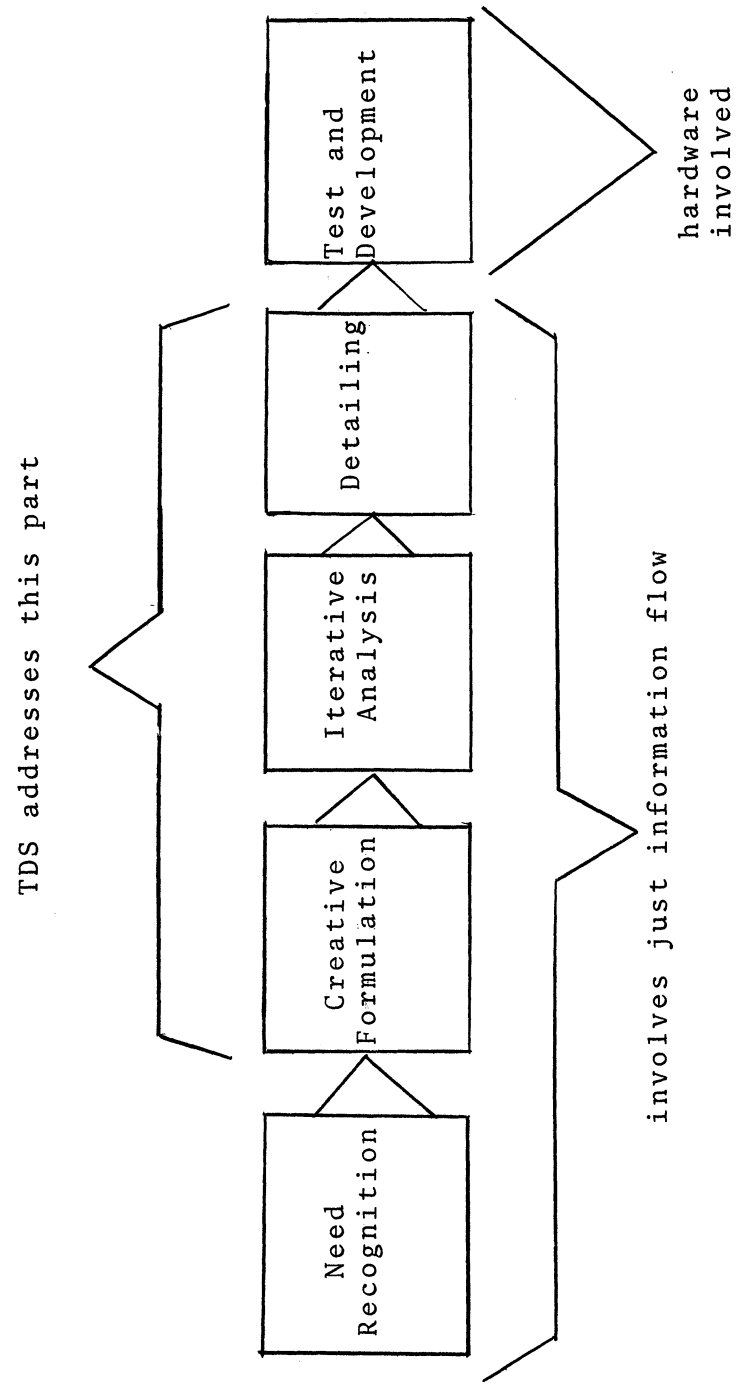


Figure 2.2. The Customarily Conveived Design Process.

arily conceived design process as depicted in Figure 2.2.

The design process is seen basically as a sequential set of five steps. There usually exist feedback loops in this process between some or all of the steps. The correlations between some of the designer's information inputs and the steps in the design process are fairly obvious. For instance, the first pass through Need Recognition generally consists of specifications from the designer's environment. However, the usual case is that some time before arriving at the Test and Development stage a truer picture of the "real" need is achieved and significant feedback takes place.

It can be argued that "need recognition" is not a step in the design process. The basis would be: no need—no problem. However, in most cases the stated need does not describe the crux of the problem. It is the task of the designer to spell out the real problem that must be solved before an overall solution can be sought.

In recognizing the essential need of a design problem, some process goes on in the designer's mind, and is a function of his understanding of the physical world, his intuition, etc. Weighing alternative problems as being the "key" ones is done "in the control volume" too, just as conceiving them was. He might decide which one(s) is (are) significant with the aid of a pencil and paper or a computer, but these latter tasks are immediately "analysis." So, for this step of "need recognition," what goes on in the control volume with respect to information manipulation is an unknown set of "mental processes";

however, it can be said that any facts involved were previously input information.

Creative formulation is by definition a process internal to the control volume. However, there is information flow through the working interface when candidate concepts are to be related to the environment for the construction of models.

Shifting attention to the "detailing" step, one can see that if creative formulation can be considered analogous to layout drawing while detailing is considered analogous to detail drawing, then what is actually involved here are varying degrees of conceptual refinement.

The word "iterative" in the iterative analysis step implies the feedback-causing nature of the decisions involved. In broadest terms this can mean changing variables in a mathematical model, varying the parameters of a real physical system, or manipulating model topologies. Changing the variables in a mathematical model further implies the notion of information retrieval. This latter consideration is generally ommitted from discussions of the design process. However, as was mentioned in section 1.2.3, the designer, in assigning values to variables, usually exercises discrimination on available data as a function of his current knowledge of the physical world. The key word of course is "available."

The last step of the design process to be considered is test and development. This input to the designer is essentially the same as the real physical models discussed above, except that there are no scale effects since the scale is unity.

In summary for this section, some of the customarily conceived steps of the design process have little or no information flow through the working interface. In fact, as far as information flow between the designer and his environment is concerned, it exists in two situations. Namely, for: (1) conveying varying degrees of conceptual refinement and the associated iterative analyses, and (2) interaction with real physical models.

2.2 The Meaning of "Structure" Relative to Design Information

As pointed out in the previous section, processes have structure. Hence, for delimiting the structure of design information one logically looks at the design information that relates to processing. As shown in Figure 2.3, information interchange is involved with mathematical models of physical systems, and with physical models inasmuch as real variables are retrieved to become model parameters.

The classes of information that are brought to bear on a design problem are shown in Figure 2.3 by the dotted enclosures. Those classes within envelopes A and C are the structural elements of design information that relate to processing. (Envelope B will be discussed in Chapter 4.)

When these classes "communicate" among themselves, the information flow dictates the arcs of a network whose nodes are these classes. Consequently, the structure of design information is as shown in Figure 2.4. These relevant classes of design information are connected by lines indicating the dependence of one class on another in order to be of use to

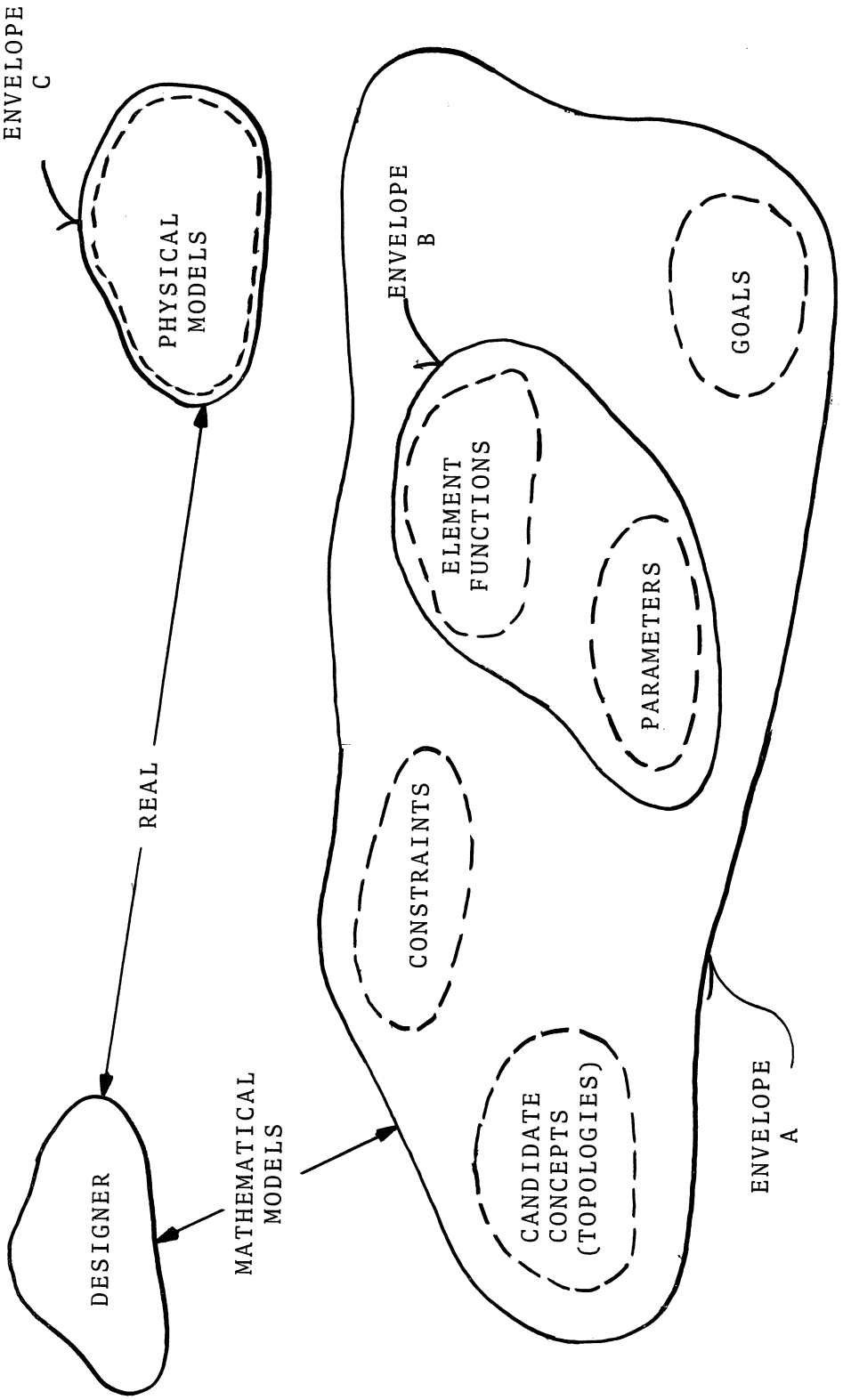


Figure 2.3. Categories of Information Flow, and the Classes of Information in the Design Process.

the designer. The arrows should be interpreted as meaning "the possibility of using or achieving the class at the tail of the arrow, depends on the existence and character of the class at the head of the arrow."

In Chapter 4 the communication of this network will be analyzed, as the information processing operations that work on these classes of information are structured.

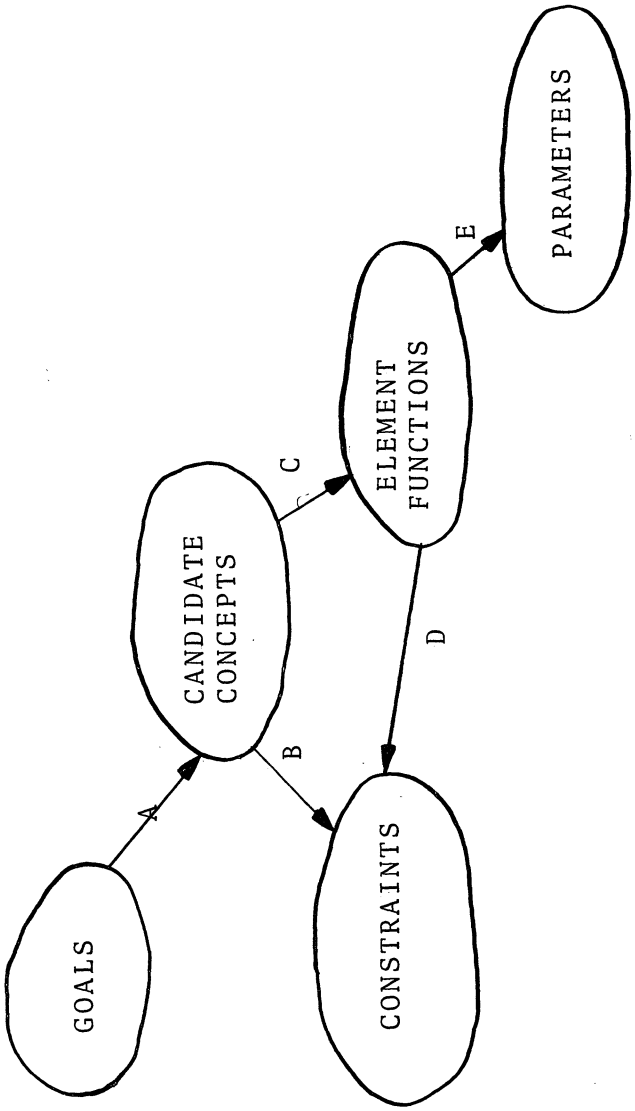


Figure 2.4. The Structure of Engineering Design Information

CHAPTER 3

COMPUTER AUGMENTATION OF DECISION MAKING FOR DESIGN

In the past ten years, the computer has brought about profound changes in the way engineering analysis and design are performed. In analysis, the computer has become a vital adjunct to theory. Theory establishes the foundation of the subject matter; computation provides clarity, depth, and insight. However, theory is not the only essential ingredient for analysis. In order to bridge the gap between theory and practice, an algorithm or approach is needed. Indeed, the very process of devising an algorithm enables the engineer to understand more fully the analytical bases of his problem. In design, the computer has thus far made trial and error and various optimization procedures practicable. It is one intention of this work to show that the computer can make a large additional contribution by making graphic input of problem topologies practicable.

It has often been said that the computer age will do for man's mind what the industrial revolution did for his muscle. The fundamental point here is not that machines lightened man's work *per se*, but machines were able to take over mundane tasks and allow man to use his muscle for other new, and more sophisticated endeavors. Further, machines have provided men with very high-leverage transducers for interacting with material objects. And, reflecting again for a moment on any real world system, there are two types of inputs: those known as materials or objects, and information. Now, if the

computer is going to do for man's mind what the industrial revolution did for his muscle, actually what is going to happen? It is proposed here that first, the immense retrievable storage capacity of the computer will provide the very high leverage needed to process the quantities of information associated with a complex system. Further, graphic, voice, and "high level" language man-computer communication will relieve the designer of mundane communication tasks by decreasing the excess information (see section 3.2.1 and Appendix 5) required to convey his messages. Thereby, the designer is left with more time per unit working hour to consider the true problems associated with his system design. Hence, the part that the computer will play begins when the designer wants to communicate with his environment, and wants to have very low excess information in his message, yet he wants to discuss a very complex system. The computer will then supply the storage capacity and it will also supply a method for communicating at a "high level."

As pointed out in section 1.2.2, computer augmentation of decision making is a combination of teaching machine, computational, and information storage and retrieval characteristics. These capabilities, when properly blended in an interactive computer-based system, provide a new dimension to the designer's environment which allows him greater degrees of freedom for the act of decision making.

A difficulty arises when one wants to apply the power of digital computers to design problems in this context. That

difficulty is: getting engineers and scientists who are trained in design, analysis or other specialties to use computer aids in their every-day work. The major contributor to this usage barrier is the typical requirement that the user be fairly well trained in computer technology. A specialist in one field cannot be expected to divert effort from the pursuit of his primary work to devote himself to detailed computer training. If, then, computer problem-solving aids are to be employed by non-computer specialists, how might the designer's approach to these aids be made easier? One promising answer is to use computer-controlled display consoles as the user-computer communication interface.

3.1 Developments Necessary for Effective Computer-Augmented Decision Making

Forming the basis for effective computer-augmented engineering design involves the development of:

1) computer hardware
2) operating system software
3) general-purpose user-oriented software
4) application software

Historically, the engineering designer has been well into the process of design before taking advantage of computer power. The reason for this is as follows. Once computer hardware and operating system software was available, user-oriented compilers such as FORTRAN became available. These were based in some kind of batch, or more recently, time-shared system. Engineers were glad to have the routine analysis burden lifted

from them by being able to develop application software within this framework. They have since developed many analysis routines for discrete elements in the systems that they were designing. However, to this time, there has not existed much general-purpose, user-oriented, non-discipline oriented software which comes under the consideration of item number 3 above. The reason is: it has taken the development of a time-sharing environment to allow a reasonably economical approach to both complex analysis and information retrieval. The implementation described in this paper falls in the realm of general-purpose user-oriented software in that it is for the design of discrete element systems, but is otherwise context-independent.

3.2 Some General Considerations

Many considerations related to computer augmentation of decision making are related to the problems of human factors engineering. This is of utmost importance because an interactive system, or an interactive portion of the user's environment, which frustrates or otherwise annoys the user is really of no practical value. Consequently, the following self-evident considerations are almost rules to be satisfied. First, there should be nothing that the user can do in the way of in-putting information to a computer-based system to cause the system to abort completely. Second, the system should coach the user when errors are made. Third, distraction should not be caused by slow interface response. And finally, any language used (especially its syntax) should seem natural.

3.2.1 The Notion of Excess Information

A first introduction to the notion of excess information can be made with the aid of Figures 3.1 and 3.2. In Figure 3.2 the screen of a cathode ray tube is depicted with a small square drawn approximately in the center. In Figure 3.1 is a sequence of computer instructions (DEC-338) which,to those knowledgable in the language, also conveys a small square at approximately the same position on the screen. The point here is that in both cases there is at least the following amount of information for someone familiar with the computer instructions: a one-inch square exists, approximately centered on the screen. It takes quite a bit more information on the part of the person conveying the notion of the square to the computer to convey that notion in a method as depicted in Figure 3.1. What one must know in addition to how to draw a square are the following: the notions of iteration, vector modes, data states, core locations, statement labels, scale values, point modes, sector bits, intensity levels, and several other notions—all in the realm of "computerese." Consequently, the fact that excess information is involved in the representation in Figure 3.1 seems to be intuitively obvious. This is discussed in a quantitative fashion in Appendix 5.

Following the discussion as in Appendix 5, the lower number of bits of information that would be generated by the designer when he expressed the square as in Figure 3.2, would mean that an observer is not surprised that the designer can draw a square. Finally, it seems that if the designer imparts

EXCESS INFORMATION COMPARISON

	HIGH	LOW
SQ	SC117	
	PMCSED	
	0100	
	4220	
	VEC+EDS	
	4000	
	1300	
	5300	
	0000	
	4000	
	3300	
	7300	
	4000	
	JMP SQ+4	

Figure 3.1. Computer Code to Convey a Small Square that will be Displayed on a Cathode Ray Tube.

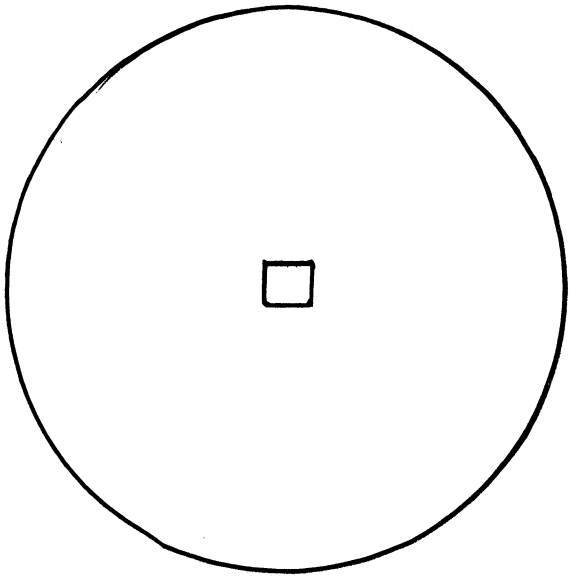


Figure 3.2. A Small Square Drawn on the Face of a Cathode Ray Tube.

more information to an observer by conveying the idea of a square in computer instructions, which means he has conveyed more notions most of which were not expected of him, then he has put more effort into getting his idea across. Hence he has exerted greater effort by conveying his idea via Figure 3.1 than by using the "higher level" language in Figure 3.2.

In Figure 3.3, two of the possible communication channels are shown from a designer to the central processor of his computer-based environment. The idea here is that the engineering designer, functioning with the proposed Terminal Design System, bases his human response on his reaction to the information channel that links him with the screen.

The designer need not be concerned with specifying the complex computer operations performed as a result of his picture language statements. This internal detail is handled by the "aiding programs" existing in the computer. The designer need be concerned solely with specifying a logical sequence of operations within the framework of these programs. To do this he need only know how to use a simple picture language which, since it can be oriented toward his specialty, can generally be easily learned.

Consequently, a channel requiring less excess information does exist if he can communicate a figure to the computer by merely drawing it. It is acknowledged that this does not cut down the information content of a message which goes from the designer to the central processing unit (CPU). The "aiding programs" interpret the inputted graphical message and supple-

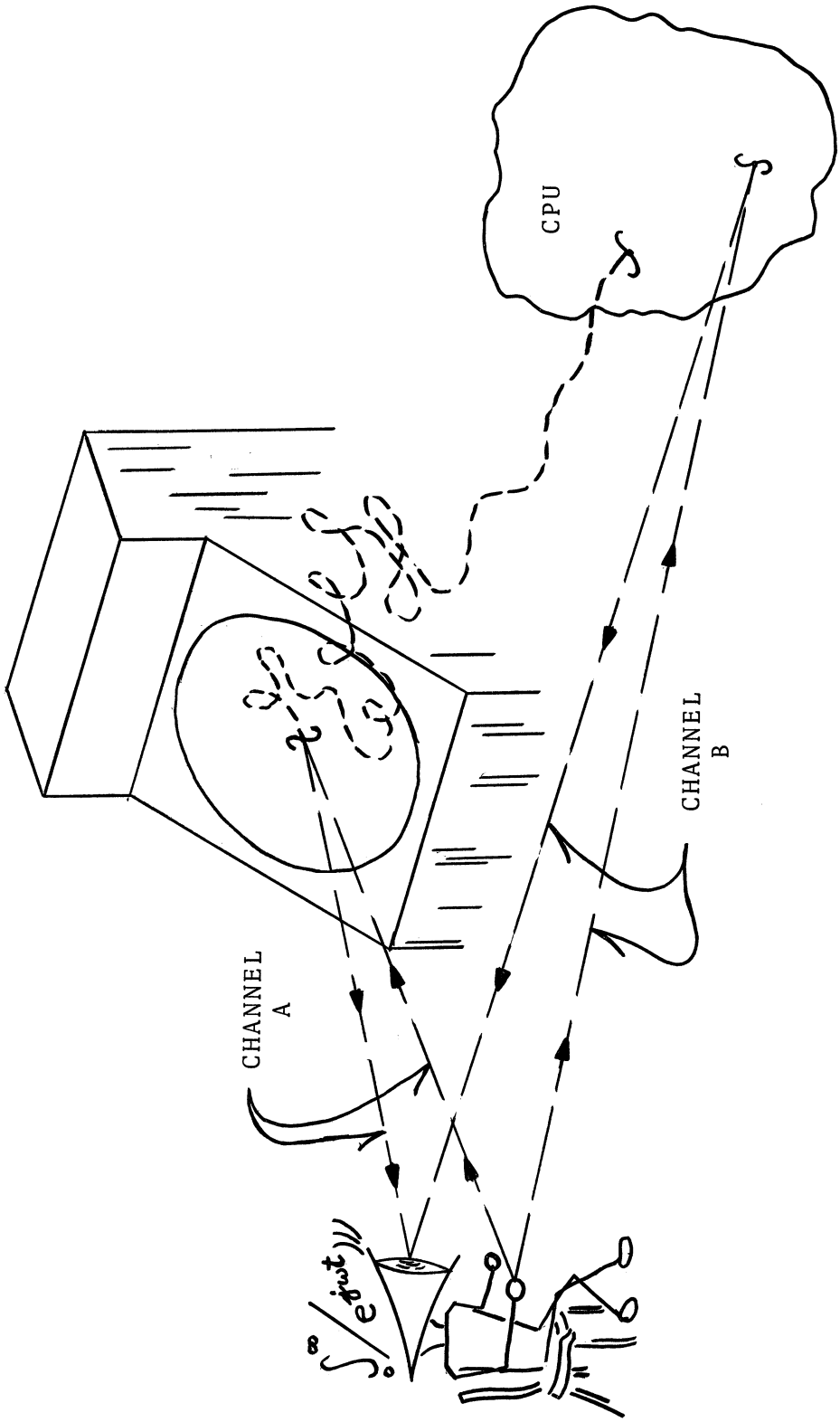


Figure 3.3. Two of the possible Communication Channels between a Designer and the CPU of a Computer-Based Environment.

ment the information to put it in a form acceptable to the CPU.

Another notion when considering excess information is the idea of definitions. A "definition" is an entity which suppresses detail. If one can allow suppression of detail, then once a designer has achieved a certain level of success in the design of a system, he can call that segment or that subsystem by merely declaring a name which he has associated with it. In general, definitions are used to suppress detail which is constant for some identifiable duration, and which is, can, and should be supplied automatically. Since the choice of which details are to be constant and thus suppressed is a personal one, it should be under control of the designer.[1]

One further notion of excess information has to do with using the highest level languages that are practical for a particular purpose. A slight amplification of this is in order. For instance, if a designer wants to employ some individual operation, but the relevant operator in the available language collectively employs a small population of individual operators of which the desired is a member, the language could then be considered at "too high a level" for his current purposes.

3.2.2 The Notion of Binding Time

In the previous section it was pointed out that definitions suppress detail. This means that some amount of infor-

---

[1] B.A. Galler and A.J. Perlis, "A Proposal for Definitions in ALGOL," Communications of the ACM, 10, 4, (April 1967) p. 204.

mation is to be considered a constant. Therefore, this information is considered to be "bound."

The essence of the design process is decision making. The more information that can be simultaneously brought to bear on an aspect of a problem, the more effective the decision will likely be. Using definitions, or binding details, allows more information to be considered. However, binding reduces flexibility. The designer wants to be as flexible as possible in order to leave ways open for attacking other problems as they arise. A similar concept is mentioned by Wilson and Wilson, and is called the "Principle of Minimum Commitment." They state the general principle as follows: "At each stage in synthesizing the design of a system, make no commitment beyond that necessary to solve the problem at hand."[2] Thus the notion of time is implied, and binding time can be interpreted as tying the concepts of designer effectiveness and designer flexibility together.

In section 7.1, being able to conveniently "un-bind" most any decision and iterate on a new set of conditions is discussed as being a very powerful characteristic of the design system described here.

3.2.3 Attention Spans

Distraction should not be caused by slow computer response. This tends to create a period of time which is not fruitful in the interactive sense. The attention of the design system user begins to wander as the attention spans re-

---

[2] Ira G. Wilson and Marthann E. Wilson, Information Computers and System Design (New York, Wiley, 1965) p. 194.

quired of him get longer. However, the actual interaction time is only one of many aspects affecting attention spans of the user. Another, which has been frequently un-treated in existing systems is the notion of cutting down the excess information required to convey an idea either from the user to the computer or in the reverse direction. The whole point of cutting down the required attention span is not, indeed, to save the user time. Rather, the point is: it seems that in the engineering design process, the less conscious effort that the designer has to bring to bear on the communication of his problem, the more free his mind apparently is to conceive of creative alternatives to the design problem at hand.

3.2.4 Kinds of Graphic and Printed Communication

Design information is conveyed by signs, and with few exceptions, the signs must be suitable for communication between human beings. These signs are one of the subjects considered in information theory. Usually, desired objects, and the operations for producing and assembling them, are described in these "designer's signs" as placed on "drawings" and/or in "instructions." Engineering drawings use their own peculiar signs and conventions that are quite unintelligable to a person without adequate training. In fact, the amount of information represented by the conventional signs and text of the engineering drawing is very difficult to determine. In the future, the ability to measure the amount of information in a designer's drawing will make it possible to compare different approaches to inputting the essential infor-

mation. Future work on this subject is discussed in section 7.2.

There are several purposes for which graphics can be employed. First, as shown in Figure 3.4, the lines indicate either entities or connections. However, these "graphics" do not convey "data" as they do in the full-scale drawing of Figure 3.5. The third purpose for graphic communication is demonstrated in Figure 3.6 where transmitting the configuration or "feeling for the layout" is the goal of the graphics.

There are other notions related to communication of graphic symbols. In relational relationships, of which a good example is an equation, one side of the equation is related by some symbol to the other side of the equation. This implies that, when treated mathematically, an implicit or explicit relationship holds. In graphic or printed communication, there is the positional relationship, a good example of which is having the characters "3 ohms" being in the proximity of what is generally conceived to be a resistance symbol. And this resistance symbol, is itself an example of an explicit relationship.

3.3 The Influence of Design System Structure

In a typical situation the designer of discrete element systems submits a system network to the computer for analysis. The results from the computer provide the designer with insight and additional information which he uses to generate a new or modified network for investigation. Also interwoven here are periods of information retrieval. Until very recently,

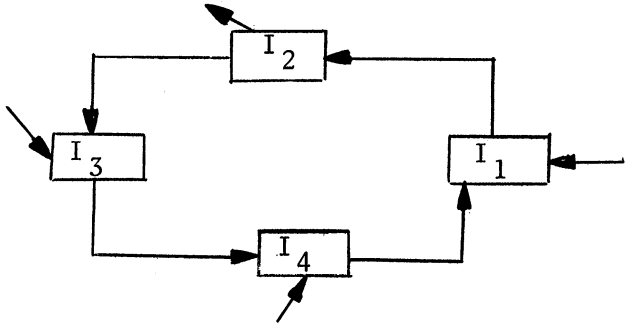


Figure 3.4. Graphics Conveying Topology.

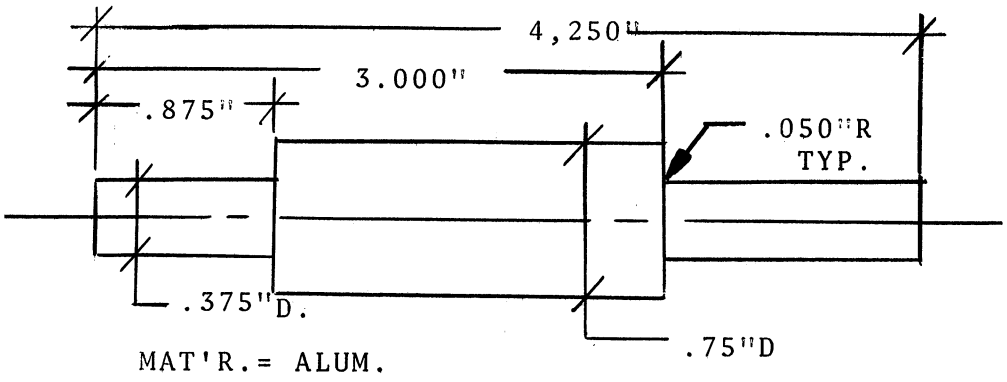


Figure 3.5. Graphics Conveying Data.

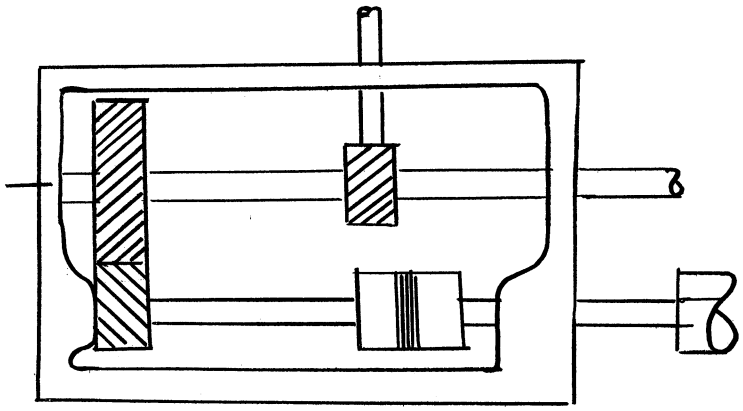


Figure 3.6. Graphics Conveying "Feeling for the Layout"

the designer had to prepare his input to the computer on punched cards or tapes, and then submit his job for a batch run the results or which became available only after an always present delay, "the turn-around time," which is typically one to two hours.

The above mode of man-computer interaction has had two major defects:

1) these formerly conventional forms of input to the computer, that is, cards or tapes prepared according to formats specified by the programs, are basically alien to the design engineer. The designer is usually plagued by input data errors until he develops a familiarity with the particular programs; sometimes, only to find out that he has been working on the wrong program. These errors due to improper input data preparation, disrupt the design iteration cycle and frustrate the designer.
2) more importantly, the delay due to a relatively long turn-around time in feeding back computational results to the designer, makes it impossible for him to try out ideas rapidly. This delay undermines the effectiveness of the iterative design process.

One factor which would help bypass these man-computer communication problems would be the use of on-line consoles which have graphical input and output (I/O) capabilities. Now, on-line graphical (I/O) capability can be implemented in a myriad of ways. The following discussions elaborate on

some of the major points involved. First, however, the man-computer display loop is depicted in Figure 3.7 below.

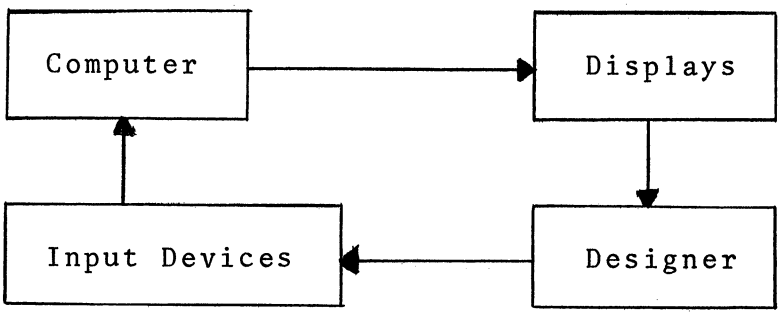


Figure 3.7 The Man-Computer Display Loop

The parts of the picture on a graphical console represent both commands to activate certain programs (for example, light button service); and, once the programs are activated, arguments or data for the programs. Of course, this principle of dual function is also true of the interactive typewriter.

3.3.1 Data Structures

Several times above, "interactive graphics" and the "interactive typewriter" have been referred to. The following discussion relates to those structural features of a design system that make the "—active" connotation a reality.

Graphic Data Structures

A graphic data structure must support both the graphic command language (light buttons and comments), and an interpretable user-generated symbolic topological representation

of the current design problem. The reason for this dual function is obvious—both are required. Many considerations are implicit in the word "interpretable." The reason that the graphic structure must be interpretable in the first place is explained by again looking at Figure 3.3. When that figure was discussed, the point was made that the net information flow from the designer to the CPU of his computer-based environment was the same on either channel A or B. However, on channel A, the designer operates with less excess information by merely expressing his graphics thoughts by drawing. The drawing is then interpreted by his environment and processed (supplemented) according to some conventions in order to convey the required information to the CPU.

The significance of the data structure relates to "interpretation" and is explained by stating some of the kinds of questions that the environment must be able to ask about the portions of a sketched problem topology. They are:

1) To what other system elements is this element topologically connected?
2) Is the relationship active or relational?
3) What direction is the information flow between instances (system elements)?
4) What is the name of this instance?
5) What are this instance's port label names?[3]
6) Which of all the defined instances are germane to

---

[3] For the definition of "port label" see Figure 4.3 and its related text.

the current problem (i.e., which ones constitute the network being analyzed)?

Operational Data Structures

Engineering systems are inherently combinations of subsystems. Because one task of the designer as pointed out in the discussion of Figure 2.3 is the suggestion of candidate concepts, the structure of any design system must allow the user to manipulate the elements under consideration. This not only means add them to and delete them from the valid problem network, but also modify their individual properties. In order to do this the data structure in which the element information is imbedded must be able to quickly answer many questions about the problem's components. For example:

1) What elements are defined?

2) What are their port labels?

3) What are the associated units?

A method of achieving this is outlined in Chapter 5, and the operators employed here are listed in Appendix 3.

Information Retrieval

Similar to the considerations above, the designer must be able to interrogate his environment about material properties, currently defined entities and value-ranges quickly, conveniently, and possibly with some sort of generalized coaching. The reason for being able to do this is that an available wealth of relevant data can enhance the performance of the designer. Over a long time period the quantity of information in a message going through an element cannot be increased.

In a design system, however, any individual user does not see the system for a long enough time span to be considered "long time." Hence, if each individual user is in essence on a short-time basis, the stored information when coupled with his input can, indeed, apparently increase the information content of the messages transmitted between the computer and the designer. Obviously, the amount by which message content can be increased would be a function of calendar time, as the library would be built up with continuous usage. Because the nature of engineering information is class oriented, (e.g., units and unit conversion), an associative data structure very naturally allows the kind of retrieval envisioned (see Appendix 1, section 3.1).

3.3.2 Computational Capacity

In Chapter 2 the point was made that design information amenable to processing is associated with mathematical models and their related analyses. These analyses are generally performed by "applications programs." Today, an abundance of these programs exists; however, they are of varied languages and formats. Furthermore, these programs have been based on a myriad of assumptions and perform to varying degrees of accuracy. All of this is to point out that flexible computational capability must be available.

Another consideration is the size of these individual analysis programs. Many of these available routines are very large. A design system requires computational flexibility that will allow arbitrary analysis routines to be employed.

This can be provided by a sufficiently general (which implies large) operating framework. What this says about the overall structural organization of a design system will be covered in Chapter 4.

3.3.3 Teaching Machine Capability

In section 1.2.3 the two connotations of "teaching machine" were explained. The way that system structure affects each of these is discussed below.

Relative to Engineering Content

In this case the message interchange is context dependent. The messages to the user have the effect of increasing his understanding of the engineering principles involved in the current problem. The idea here is that the structure in which he operates must be able to allow retrieval of user-specified classes of information. It is possible with set-theoretic operations such as "intersection" to correlate entity and unit synonyms. For an example of this see section 3.2 of Appendix 1.

System Performance

The other meaning of "teaching machine" has to do with the actual operation of the design system. The structure must allow the user and the computer-based environment to "parenthesize" comments to each other. That is, a dialog must be allowed about the system's operation—*while* the system is processing a design problem, and *without* process interruption. The principle for achieving this is discussed in the second section of Chapter 4.

Furthermore, because it has been established that a design system must be adaptive, it also follows that the allowable coaching dialog must also be adaptive. Frequently, documentation of this kind is not well done because it is a task in parallel with the system growth. Hence, initiation of changes should be done automatically.

3.3.4 Hardware

The influence of hardware can be covered in blanket fashion. The basic characteristics necessary for the implementation of a system for design are:

1) a large directly-addressable memory space. Recall that in section 3.3.2, one system design principle that gives a designer flexibility is allowing him to employ large analysis programs without having to alter them to "fit" his computer-based environment. The operation envisioned requires that a memory space many times larger than the real programs be available.
2) direct access, as opposed to sequential file capability for performance. The performance of the system described here probably could not justify the system's existence if the central computer's peripheral storage was on tapes instead of discs.
3) the ability to reproduce multiple copies of some graphic picture part. This can be done either with a subroutining capability or with large storage; however, the latter is inefficient in a limited-size peripheral

computer when all the graphics is based peripherally.

4) the facility of "picking" or selecting a graphic picture part.

CHAPTER 4

THE STRUCTURAL ORGANIZATION OF A COMPUTER-BASED DESIGN SYSTEM

This chapter explains the establishment of a structural organization (processing configuration) that will synergistically handle the design information of reticulated systems.

4.1 Considerations Which Precipitate from the Class Structure of Design Information

In Chapter 1 the notion of systems and subsystems was established. Further, any system design task can be described as: finding the entity or entities that will transform a given set of inputs into a desired set of outputs. In Chapter 2 the point was made that the classes of design information depicted in Figure 2.3 "communicate" such that the net information flow among them proceeds from a set of inputs to a desired set of output goals.

4.1.1 Inter-Class Communication

Physical systems are composed of elements which are transfer functions for forces, voltages, etc. Their corresponding analysis routines (element functions) quite obviously constitute one required class of design information. However, these element functions usually represent the characteristics of a set of physical elements. To make them unique for a physical element, parameters must be specified. Hence, another class of information is the set of possible parameters to be used with the element functions. An example of this might be a gear reducer analysis routine for which face widths, shaft diameters, speed, torque and other parameters must be specified.

The usefulness of element functions therefore depends on the existence of the required parameters. However, the mere existence of the necessary parameters is insufficient; they must exist in units compatible with the rest of the element function whether they are retrieved from storage or monitored "live."

Element functions have inputs and outputs just as physical elements do. The fact that all members of the "constraint" class are inputs, but that all inputs are not constraints alludes to the input specification problems of necessity and sufficiency. Also, the class of goals is mapped onto the element function outputs, and quite obviously, if their intersection is a null set, no solution can exist.

The remaining class of information, candidate concepts, describes the "layout" of each design problem. This class must exist for physical significance.

4.1.2 Some Specifications for the Structural Organization

Some of the classes of design information in Figure 2.3 immediately have relevance to the structural organization. First, those classes in envelope B are to be stored for use by subsequent designers. This requirement for storage capability means that a design system must have a library facility. More than this though, the "librarian" must accept additions while giving due consideration to the formats required for future element function communication.

Another needed aspect of the structural organization is a topology (candidate concept) interpreter. There must

be a facility which takes the designer's candidate concepts and interprets them for use in determining the inter-element information flow.

4.2 System Characteristics Required for Synergism

The characteristic operation of a design system must be such that the effectiveness of the user/designer is in all respects enhanced. In practice this means several things.

4.2.1 A Two-State Interface

First, at all times the designer must have absolute control of the progress of his problem solution. Two situations must be considered. The first, is when the designer is inputting new data, and the second situation is when the design system is struggling to develop a solution from the given data. Here, the user must be able to monitor the information flow with the option to interrupt at any time. Consequently, a two-state interface is required.

As a Controller

In order for a computer-based design system to offer the user preemptive rights at any time while supplying or requesting information, every message must be filtered. For example, while working in the "Library" mode, a sequence of characters such as $T (which is a command to go to the topology specification mode) is constantly being looked for. Further, in the event of trouble, an attention signal must be allowed which supercedes all previous commands and has immediate interrupt priority.

As an Interpreter

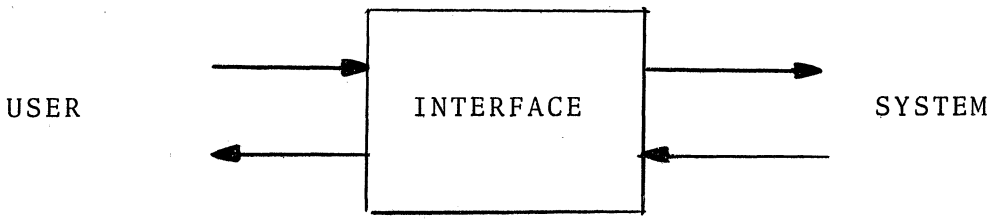
The other state for the user interface comes into use whenever the system must interpret and manipulate data. Here, the "attention" property must still exist, and the user must be able to monitor all system functions. Further, the capability of the system to interpret "alteration" commands must be available so that the user can spontaneously pose questions to the system or alter some already-specified portion of his problem.

From the above discussion it is clear that the interface must possess the properties of a transformer at certain times, and a modulator at others. This idea is shown in Figure 4.1. Borrowing from the terminology of network theory, the interface must at some times act as a two-port device and at other times as a three-port device. In essence it must be a variable multi-port device. This idea will subsequently be referred to as follows: the user interface must possess the characteristics of a Vari-port link (VPL).

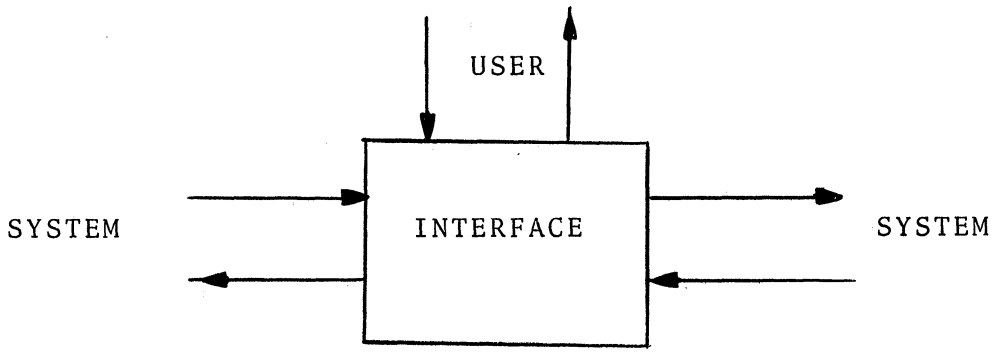
4.2.2 'Set' Orientation for All Information Classes

The second requirement for synergism is that there should be as fast a response as possible to all user requests and commands. This is in keeping with the previous discussions of the effectiveness that is lost when a designer's train of thought is broken by slow interaction.

It is well known that retrieval from an associative (set oriented) memory can be faster than list searching for many applications. The writing of payroll checks from employee



AS A TRANSFORMER



AS A MODULATOR

Figure 4.1. Interface States of a Computer-Based Design System.

personnel files would probably be faster if implemented using list searching (in this example sequential selection). Yet, if the question "How many married male employees over forty years of age subscribe to our hospitalization plan?" were to be asked, a random access method of retrieval would provide the answer quicker than list searching. The design process is such that retrieving one piece of design information gives zero information about what might have to be retrieved next. Hence, if the structure of a design system is built to operate associatively the response will probably be faster.

The consideration is: how does one represent each of the classes of design information in associative form?

An Instance Is a Relation

A mathematical relation is a set of ordered pairs. Furthermore, the graphic instance of Figure 4.3 can be represented by the relation:

$$N_{COMPR5} = \{<COMPR5, L_1>, <COMPR5, L_2>, <COMPR5, L_3>\}. \quad (4.1)$$

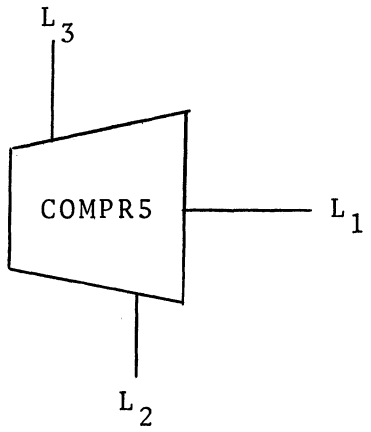


Figure 4.3 A Typical Graphic Instance

An instance can be defined as a relation the domain of which is the instance name, and the range elements of which are the port labels.

Taking this concept one step further, a system is a relation. Consider the following relation:

$$S = \{<<N_1,L_1>,<N_2,L_3>>, \ \ldots \ \}. \qquad (4.2)$$

Here, each element of each ordered pair is itself an ordered pair. In fact, each element (N_i,L_k) is a unique port label, if no two instance names in the same problem are identical. This relation S describes the entire topology of the system because connectedness is given by the pairs and directionality is given by their order.

An Associative Library

The first reason for having a library in which the information is stored in associative form is so that nested questions can be asked by the designer and quickly answered. For instance: What element functions are available that have the word HEAT in the title and have a specifyable mass flow rate input?

Another reason is that if references to the stored element functions and parameters are themselves in relational form, then set theoretic mathematical operators can be used on them. This allows quick checks for unit format and conversion. It also readily allows mapping of instances onto their proposed element functions to check for compatibility.

Finally, with associative storage, high-level user-directed correction and editing routines can be written (see

Appendix 1) that allow the designer to alter material already processed by the design system.

4.2.3 An Adaptive Capability

The final required characteristic is that the system be adaptive. In Chapter 2 it was mentioned that this very research has been conducted according to the design approach outlined there. Quite obviously, the possibility for feedback exists for this problem as it does for others. Given that, it must then be possible to have the system adapt to new user requirements. This means that the embryo structural organization must be capable of restructuring itself to accommodate future user needs.

Furthermore, given that a system is adaptive, because it has teaching machine capabilities its documentation must also be adaptive. This is difficult because these are essentially parallel tasks that must be done sequentially.

4.3 The Structural Organization

Based on the above, the basic elements of the structural organization, as shown in Figure 4.4, are:

1) Vari-port Link (VPL) - to help make possible synergistic communication
2) Library - for the storage of element functions and parameters.
3) Topology Interpreter (T/I) - for interpreting a possible method of information processing from a given problem network.

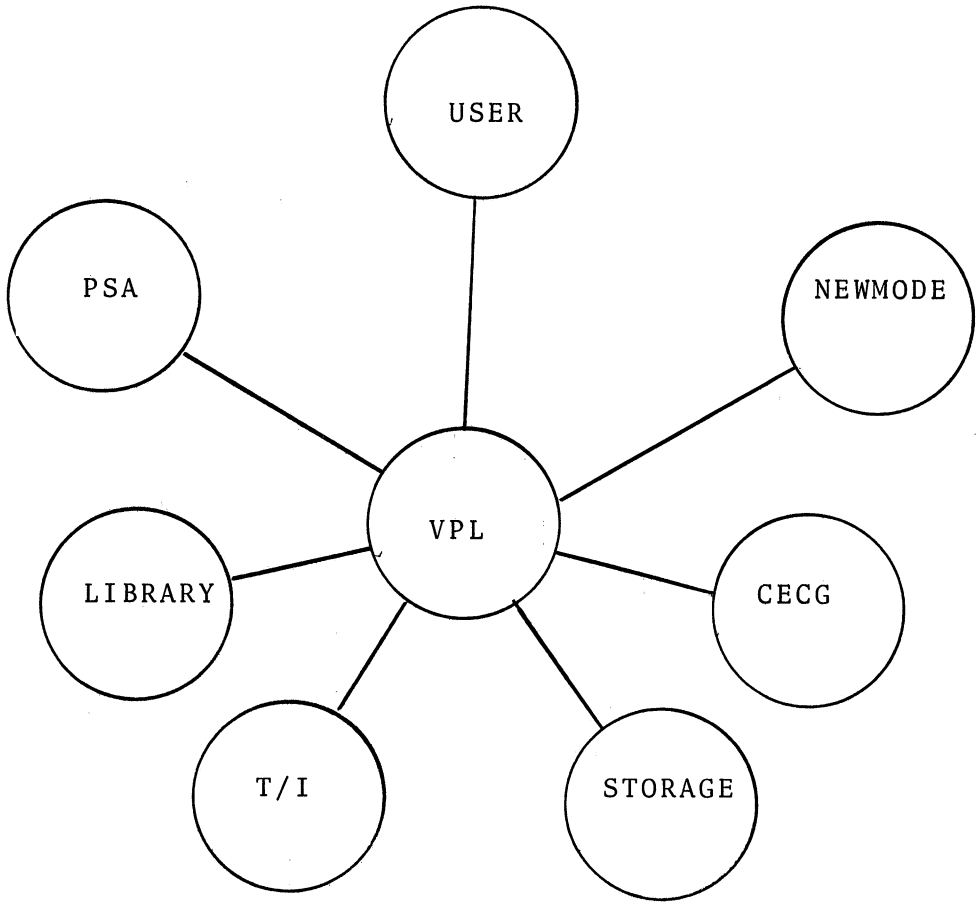


Figure 4.4. The Structural Organization.

4) Problem Staging Area (PSA) - for construction of problem solution instructions.

5) Storage

6) Coaching and Error Comment Generator (CECG) - for providing "teaching" type assistance.

7) Newmode - for adding new user-specified modes of operation.

4.4 Using This Organization to Process Design Information

Processing the information associated with the system networks of mathematical models means considering networks of information flow.

4.4.1 Description of the Given Data

Given a system relation of the form of Equation 4.2, an information network of the type shown in Figure 4.5 can be derived. Regardless of the nature of the real physical system being modeled, the corresponding information net of the

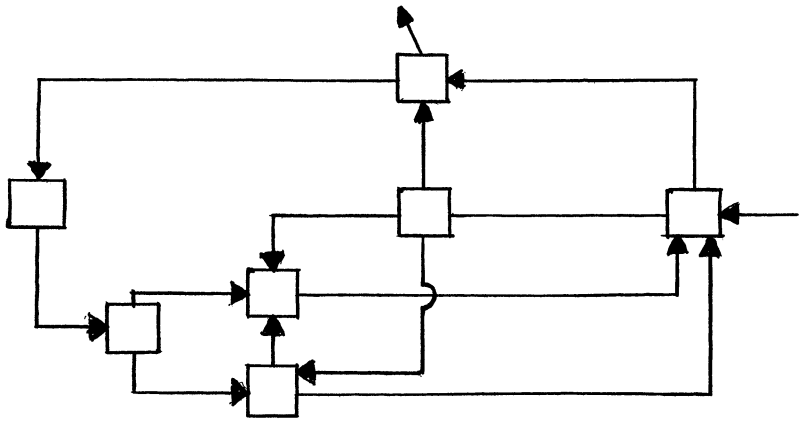


Figure 4.5 A Problem Information Network

mathematical model is a directed graph, which can be represented as indicated by the arrows of Figure 4.5. This is referred to as the topology of a design problem.

4.4.2 Interpreting the Given Data

It is fairly easy to conceive of an information network such as the one shown in Figure 4.5 being derived from a system relation. At this point, element functions have already been made to correspond to the physical system elements (this is accomplished in the implementation discussed here as shown in Appendix 6). This network is ready to have a possible processing method interpreted from it. To begin processing, the set of nodes that can be starting points must be identified. Further, the remaining nodes or element functions must then be ordered for processing. This leads to the following.

*** Definition 4.1

Given the non-empty sets B,C --- A(B,C) is an algorithm from B to C if and only if it is of the following form:

$$A(B,C) = \{\langle x,y\rangle : (x \in B \wedge y \in C) \;\&$$
$$\langle x,y\rangle \in A \wedge \langle x,z\rangle \in A \rightarrow y=z\}$$

Note that this formulation says the following:

1) The domain of A is B, which is a set of "input-sets" (each input-set is an element of B and has an associated element in C).
2) If the null set is an element of B (i.e., if $\emptyset \in B$),

then it has an associated element y in C (i.e.,yeC). This allows an algorithm which will simulate an infinite source.

3) If x$\notin$B, then x as an input-set is meaningless, and the notion of "specified" can be defined as follows.

*** Definition 4.2

An input-set is <u>specified</u> relative to A if and only if it is an element of the domain of A.

As an example of the above, consider the mathematical operator SIN as an algorithm. If its domain B is the set of real numbers R, then

$$SIN : R \rightarrow R$$

x is specified relative to "SIN" if and only if it is a real number.

*** Definition 4.3

An input-set can be <u>processed</u> by A if and only if it is specified relative to A.

*** Definition 4.4

An <u>element function</u> is a set containing a single ordered pair {<B,A>}, where B is any input-set and A is an algorithm.

*** Definition 4.5

An element function Q is <u>parameterized</u> if and only if its input-set is an element of the domain of its algorithm. [i.e., Q is parameterized iff $\mathcal{D}(Q) \in \mathcal{D}(R(Q))$,

where $\mathcal{D}(A) = \{x: (\exists y)(<x,y>\in A)\}$,

and $R(A) = \{y: (\exists x)(<x,y>\in A)\}]$

*** Definition 4.6

To <u>execute</u> an algorithm A is to have it process an input-set.

*** Definition 4.7

The list of parameterized element functions of an information network ordered for execution is called the priority list (P-list).

*** Proposition 4.1

An element function $G = \{<B,A>\}$ can be executed if and only if it is parameterized.

PROOF:

1) If an element function is not parameterized, then by Definition 4.5 its input-set must not be an element of the domain of A. Hence, by Definition 4.2 the input-set is not specified, by Definition 4.3 it cannot be processed, and by Definition 4.6 the algorithm cannot be executed.
2) Because the domain of an element function either is or is not a member of the domain of its range, an element function can be executed if and only if it is parameterized.

If every element function (node) in the network has at least one input which is "known" only as a result of the execution of another element function, then at some point in

the network at least one node is required to have all of its inputs of this type known (possibly by estimation) so that execution can begin. This idea leads to the following definition.

*** Definition 4.8

A trial value is that estimated value which is assigned by the designer to a member of an input-set when that member is also a member of an "output-set" of another element function in the network.

The idea of developing a computer-based design system is to enable the designer to observe "system" effects while treating more component interactions quantitatively. However, if he has to estimate or guess many of these interactions, the design system is no improvement over conventional interactive analysis. Consequently, it seems that the interpretation for a possible processing scheme is dependent upon trial values.

The Role of Trial Values

*** Theorem 4.1

The minimum number of trial values required for an acyclic network is zero.

1) By the definition of "acyclic," no arc progression exists which terminates on itself; hence, the first parameterized element function of each arc progression satisfies the hypothesis of Proposition 4.1.
2) From the definition of an arc progression, the k^{th} element function is reached by exiting from the $(k-1)^{th}$ element function; hence, no other element in an arc progression of parameterized element functions will violate the hypothesis of Proposition 4.1.
3) Therefore, with the first elements of each arc progression completely specified, and subsequent elements dependent only on these first elements, no trial values are required.

*** Theorem 4.2

The minimum number of trial values required for a non-acyclic network is one.

1) By definition, a cycle is an arc progression whose beginning and ending vertices are coincident.
2) With no trial values, then there exists at least one cycle in which all elements violate the hypothesis of Proposition 4.1, because each element is dependent on the previous one in the cycle for at least one input.
3) If there were no cycles the graph would be acyclic,

but with the minimum of one cycle, the minimum number of trial values is one because any element in the cycle to which the trial value is applied is completely specified thus making the rest of the cycle an arc progression of element functions which satisfy the hypothesis of Proposition 4.1.

4) Hence, the minimum number of trial values required for a non-acyclic network is one.

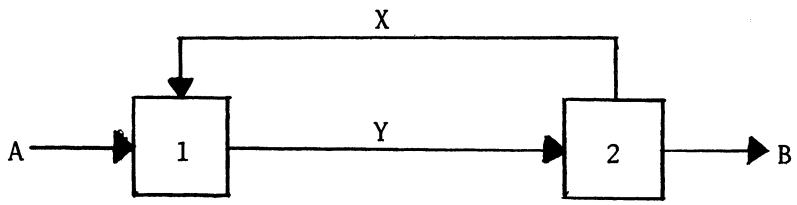


Figure 4.6 A Non-Acyclic Information Network

*** Definition 4.10

An elementary cycle is a cycle in which while traversing the cycle each vertex is encountered only once (except, of course, the initial and terminal vertices which are coincident).

*** Definition 4.11

Two cycles are disjoint if and only if they are without common arcs.

*** Definition 4.12

A cycle is open when one of its arcs is cut thus making it an arc progression with no common vertices.

*** Theorem 4.3

The minimum number of trial values required to form a P-list is equal to the number of disjoint elementary cycles in the network.[1]

PROOF:

1) If the network is acyclic, then by Theorem 4.1 the number of trial values equals the number of cycles, namely zero.

2) Consider a network having n disjoint elementary cycles. With all of these cycles open the graph is acyclic and we have the case above. If n-1 cycles are opened using trial values, then there must still exist exactly one cycle in the network. In this remaining cycle, none of its elements can be a starting point for processing because they all violate the hypothesis of Proposition 4.1. However, by cutting one arc of this cycle and parameterizing the first element function of the arc progression with a trial value, this cycle too can be ordered for processing. Hence, total number of trial values required is n, the number

---

[1] This Theorem has been discussed with Professor Frank Harary, Mathematics Department, The University of Michigan, and he pointed out that a stronger theorem has been proved in another form.

The Theorem: If a digraph is strongly connected, with p nodes and q arcs, the number of arcs that must be snipped to make it acyclic is: q - p + 1.

For the proof of the above theorem see: Frank Harary, Robert Z. Norman, and Dorwin Cartwright, Structural Models: An Introduction to the Theory of Directed Graphs (New York, Wiley, 1965).

of disjoint elementary cycles in the network.

*** Theorem 4.4

A unique ordering for the P-list can exist only if the number of trial values is a minimum.

PROOF:

1) Let the minimum number of trial values required be one (1).
2) Let two (2) trial values, x and y, be assigned to two (2) element functions.
3) If one would have sufficed, and two elements are now completely specified, either of these element functions by the hypothesis of Proposition 4.1 can be the starting point for processing.
4) Hence, with neither one depending on the other, no unique ordering exists.

The Interpretation Algorithm

Interpretation of the given data for a possible processing scheme, or P-list, is totally dependent on the existence of sufficient trial values if the network is cyclic. The following algorithm will analyze a system relation to see if a P-list can be derived.

1) From the set of instance relations for the system under consideration retrieve all instance relations which have a trial value input port, and all those which are parameterized. If one or more instance relations are retrieved, put it (them) on the "next"

list, if not, go back to the designer.

2) If a "trial value instance" does exist, and if all of its input ports are specified, remove it from the "next" list and put it on the P-list, otherwise put a parameterized instance on the P-list.
3) Put all of the instances to which that P-list entry provides information on the "next" list if they have not already been put on.
4) Pick the first item from the "next" list and see if it is either parameterized, or the ports from which it gets its information are the outputs of instances already on the P-list. If so, put it on the P-list, perform (3), and remove it from the "next" list. If not, try the next item on the "next" list.
5) Continue (4) until the "next" list is empty and all instances are ordered on the P-list. Then return to the designer—indicating success.
6) If an item or items cannot be put on the P-list, which is signified by a pass through the "next" list that results in no additions to the P-list, then return to the designer.

As an example, consider the information network of Figure 4.7, where "P's" indicate designer-specified parameters, and "T's" indicate designer-specified trial values. Table 4.1 shows the "next" list and P-list entries that correspond to the steps of the Interpretation Algorithm.

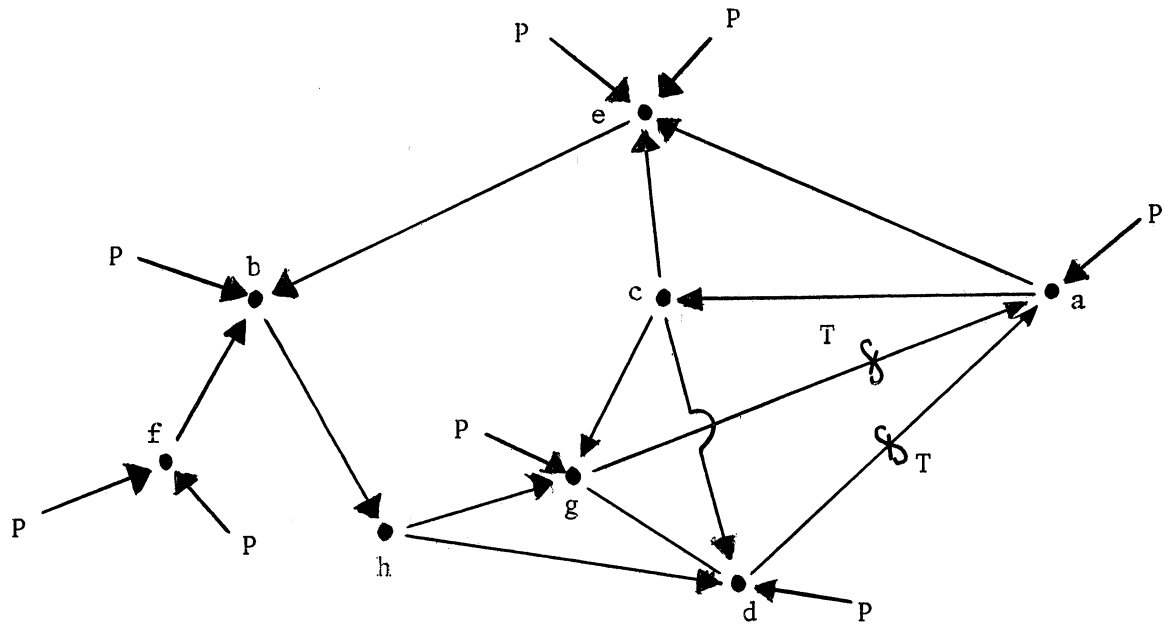


Figure 4.7 An Example Information Network

Step No.	"Next" List	Status of P-List
1	a,f	(empty)
2	~~a~~,f	a
3	~~a~~,f,e,c	a
4	~~a~~,~~f~~,e,c,b	a,f
4	~~a~~,~~f~~,e,~~c~~,b,g,d	a,f,c
5,4,3	~~a~~,~~f~~,~~e~~,~~c~~,b,g,d	a,f,c,e
5,4,3	~~a~~,~~f~~,~~e~~,~~c~~,~~b~~,g,d,h	a,f,c,e,b
5,4,3	~~a~~,~~f~~,~~e~~,~~c~~,~~b~~,g,d,~~h~~	a,f,c,e,b,h
5,4,3	~~a~~,~~f~~,~~e~~,~~c~~,~~b~~,g,~~d~~,~~h~~	a,f,c,e,b,h,d
5,4,3	~~a~~,~~f~~,~~e~~,~~c~~,~~b~~,~~g~~,~~d~~,~~h~~	a,f,c,e,b,h,d,g

Table 4.1 "Next" List and P-List Entries for an Example Implementation of the Interpretation Algorithm

The Interpretation Theorem

From the previous algorithm, the following theorem immediately follows without proof.

*** Theorem 4.6

Given a system relation with attributes (i.e., trial values, parameters, inputs, outputs), there exists an algorithm which will reveal whether or not a processing scheme exists for the given data.

*** Corollary 4.1

System relations can be processed (the network solved) if and only if a P-list can be derived from the given data.

Keeping in mind that the designer should have a computer-based system which will allow him to consider his problem per se, and not what is required to enter it into the computer, the following axiom is relevant.

*** Axiom 4.1

As the number of trial values required to begin processing increases, the number of decisions that the designer must make increases. Hence, if any of the trial values that a user specifies is not required, some effort is being wasted in addition to the distraction involved.

Note that because the system has a designer in the feedback loop, nothing is said about when to terminate a trial

solution. Depending on the current purposes of the designer, terminating processing is left completely to his discretion.

*** Proposition 4.2

Considering other variables as temporarily fixed, the designer must make fewer decisions to begin a solution if the P-list employed is uniquely ordered.

PROOF:

1) By Theorem 4.4, the P-list can be uniquely ordered only if the number of trial values is a minimum.
2) By Axiom 4.1, the number of decisions to be made increases with the number of trial values required.
3) Hence, if the P-list is uniquely ordered, considering other variables temporarily fixed, the designer makes fewer decisions to begin a solution.

In Chapter 7 this work is again discussed in terms of "Conclusions and Open Problems." The current Interpretation Algorithm does not yield a P-list based on a minimum number of trial values because the number of trial values is "dictated by" the designer not "suggested to" him as envisioned for the NEW BUILD.

The Model

The model or template which determines the necessary and sufficient specification for the processing of an information network is the n-element P-list and the algorithm to derive it, where n corresponds to the number of element functions in the problem topology.

One can imagine that even a unique P-list might well represent a maximal amount of computation; however, this design system was conceived to reduce the effort required of the designer, not the effort required of the computer.

CHAPTER 5

THIS DESIGN SYSTEM (TDS)

The goal of this chapter is to explain how TDS operates conceptually. How to use the existing prototype is covered in Appendix 1, "A User's Guide to TDS." The details of the implementation are covered in Appendix 3.

5.1 The Structural Organization

Chapter 3 established the idea that a computer-based design system should enable the designer to consider more component interactions quantitatively than previously possible, and do this with less effort. Further, while the computer has been used for interactive analysis, another connotation of "analysis" relates to the "interpretation" of input information. This interpretation is to reduce the effort required, and distraction caused by, methods of data input with high levels of excess information. "Least excess information" was also discussed with respect to several other forms of design information. Hence, the structural organization of TDS has "interaction with least excess information" as its primary design objective.

5.1.1 An External View

The "external" view of the structural organization of TDS does not mean the physical appearance of the implementation. Rather, the meaning is closer to what is conventionally called a flow chart. However, Figure 5.1, while it does give the external view of TDS' structural organization, is

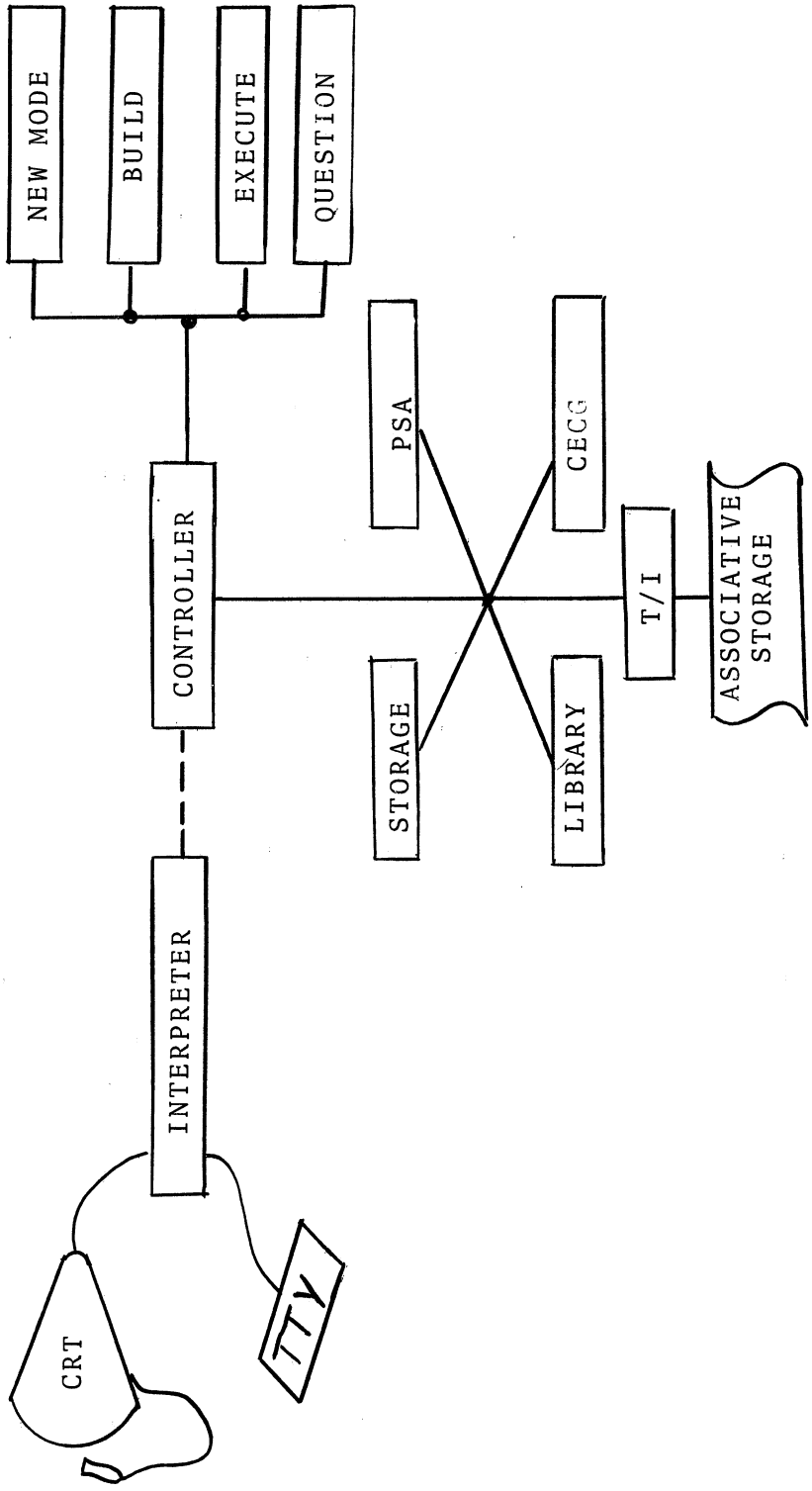


Figure 5.1. The External View of TDS' Structural Organization.

actually a "macro-component" diagram. It is not a flow chart in the conventional sense of the word because the boxes connote tasks and not functions. The similarity between Figures 5.1 and 4.4 is more than a coincidence.

5.1.2 An Internal View

The "internal" view could more aptly be called the menu of the division of labor. Following are the descriptions of the tasks performed by the major components.

The Interface Executive (See Appendix 3 - SEL/338)

This component is the operating support device for the interface. It provides all services for input-output devices such as the teletype, data communications link, light pen, etc. Further, it queues tasks so that all jobs are assured of completion in their proper priority. Probably its most important service is providing basic operators that build an interpretable graphics data structure. It is this data structure which allows a designer to have a problem topology interpreted by the "aiding programs" referred to in Chapter 3.

The Interface Support (See Appendix 3 - TDS/338)

As far as graphics is concerned, this component constitutes the "aiding programs." The service provided here first enables the designer to define graphically (in free format) system topologies, including names, and then interprets the topology so that the implied analyses may subsequently be performed. Any time that this component is in service it performs many bookkeeping tasks that enable the designer to conveniently manipulate graphical entities.

When acting as a modulator, the controller keeps tabs on the library, the problem topology and the stored information as they are used by the topology interpreter to develop a possible method for processing. Essentially, the controller operates as a modulator in the "Build" mode.

5.2 Implementation Basics

Certain basic principles have been followed in the development and implementation of TDS. An outline of the importance of each of these follows.

5.2.1 Of the Interface

Task Scheduling (with I/O Device Allocation)

A synergistic man-computer system must allow either party to initiate a request or command (a "task") at any time. If the system disallowed the initiation of a new task until a former one was finished one or both of the following undesirable situations could occur: (1) the designer might be forced to wait, or (2) the central computer may be unable to convey an answer to the designer until a prior task is completed.

The most desirable effect of task scheduling (or time-sharing between tasks) is that the user always has command, and the mundane tasks are performed more or less transparently to him.

High Speed Tracking

If a light pen is used, all light pen tasks (graphics) rely somehow on the tracking services provided by the interface executive. For drawing to be natural, the tracking must be

very fast and provide the "location" data required for drawing system networks. The design criteria must not be for the "average" man, but for the user who requires the greatest speed. Fast tracking implies that the speed and versatility of the service are worth the increased size and complexity. Of course, this assumes the existence of a high speed tracking service (measured in this implementation at approximately 100in/sec).

Operator-Interpretable Graphics Data Structure

When all the graphics is in the peripheral machine there are three good reasons for having a graphics data structure which has a canonical form.[1]

First, only a small number of operators is then required to dynamically operate on the data structure.

Second, it is possible to design operators to "ask" graphics questions such as "to what other instance are you attached?"

This means that system networks can have their elements readily altered, traded, and interpreted for connectedness and directionality.

Third, the peripheral machine is of a limited size. Yet, it is desired that the designer be able to manipulate his designs. Hence, unless machine capacity is exceeded, there is some "used" "scratch" area existing. An interpretable data structure allows an easily employed housekeeping operation to be in operation continually, insuring that only "good" graphics is retained.

---

[1] See Appendix 3.1.

5.2.2 Of the Central Computer

The Controller

The importance of having a point through which all communication flows cannot be overemphasized. By following this principle it is possible to screen every inter-component message. This seems natural when designing the "transformer" mode, but must be forced so that a "modulator" mode exists.

NEWMODE

The adaptation of the design system configuration to user's needs is one of many characteristics required to insure that the original configuration does not "constrain" user solutions. The fundamental property required is a file-oriented operating system. This not only means that files or sections of them can be executed, but that they can read and write on, and direct the execution of each other. With the capability of being able to write on each other, the future connections to the system are "system specified." This means that a user cannot extend the system incorrectly. However, the content of the extensions is up to the user, so in a sense he can attempt to add anything.

Associative Storage

The reason for storing design data associatively is so that random access retrieval will be reasonably economical. For example, engineering constants and their units are particularly suited to associative storage, as is the documentation of analysis programs. Further, if the question arises as to what elements are connected to the output of a

particular element, the answer is simply the range of the set of output connections.

Necessary and Sufficient Check

The need for this check is obvious in a sense, but the location of the check in the sequence of design events is important. It is undesirable to let the designer think that a network is ready for execution and then have him receive major operating error statements. He should have the specification checked (as described in section 5.1.2 under "The Central Computer") as soon as he tries to use his input design information. In TDS, the specification is checked any time that a match is to be made between some topology and the corresponding analysis programs.

Miscellaneous Source Languages

The idea of saving the designer work is greatly extended if the many currently existing analysis programs can at least be tried by him. However, they are generally in different languages, formats and with incompatible units. Therefore, the design system must have unit conversion, a generalized format template, and allow those languages whose compilers are available.

5.3 A Fundamental User Consideration

The analysis capabilities of this design system interpret network nodes as element functions. The design could have been based on network arcs being interpreted as the element functions. This fact must be borne in mind when problem

topologies are being described to TDS. For example, in a parallel electrical network, the passive and active elements have historically been represented graphically as the arcs. The "nodes" in these representations are locations at which the "flow" divides. In this design system, information flows. The computer cannot be expected to know how to divide a stream of information. Hence, problems must be formulated so that network nodes are element functions. Kirchoff's laws hold for information networks just as they do for electrical, mechanical and hydraulic networks.

CHAPTER 6

SYSTEM DESIGN IN AN INTERPRETIVE ENVIRONMENT

> ... in theory-construction one is concerned with information and then knowledge; in design one is concerned with information and then action."[1]
>
> Joseph Esherick

The first two sections of this chapter discuss the performance of the prototype system. First, the way in which it interacts with the designer to solve some problems, and then some remarks on its apparent effectiveness. The chapter closes with some points about the significance of the work to both engineering educators and engineers in industry.

6.1 Commented Examples of TDS' Implementation

The following examples have been selected to demonstrate several points. First, the disciplines represented are varied, demonstrating that This Design System is context free. The only common characteristic among the systems is that they are all discrete element, or reticulated. Second, the systems represented are, in reality, series, parallel or "mixed" networks. Third, the formats of the analysis programs used are varied. Finally, these examples "use" the library to varying degrees.

One final note should be made concerning these examples. Although each individual analysis is almost trivial,

---

[1] Joseph Esherick, "Problems of the Design of a Design System," Conference on Design Methods (New York, Pergamon Press, 1963) p. 79.

the solution of these problems as "networks" by other means could be very laborious.

6.1.1 Design of a Simple Refrigerator

In this problem, the horsepower required to drive a compressor is unknown. The compressor is part of a simple refrigeration cycle as shown in Figure 6.1. One of the graphic instances (network elements), the compressor, is shown in Figure 6.2 just as it was defined graphically.

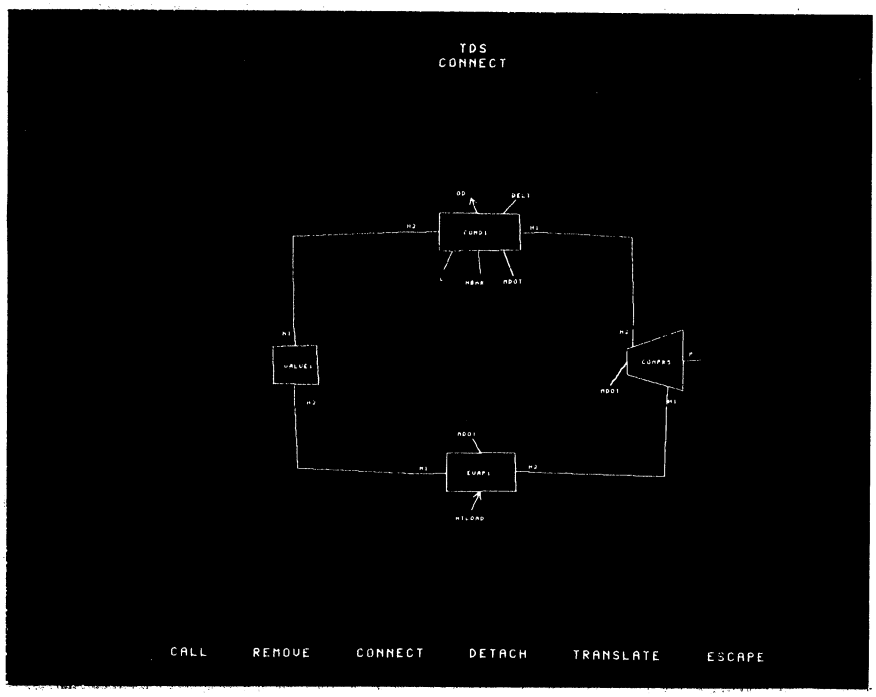


Figure 6.1 A Simple Refrigeration Cycle

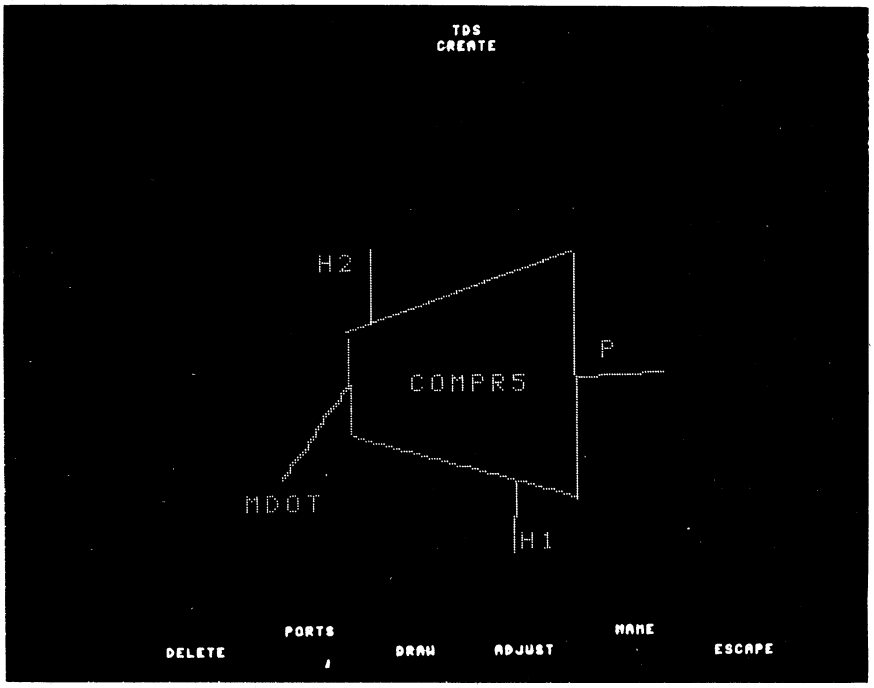


Figure 6.2 The Graphical Definition of the Compresso

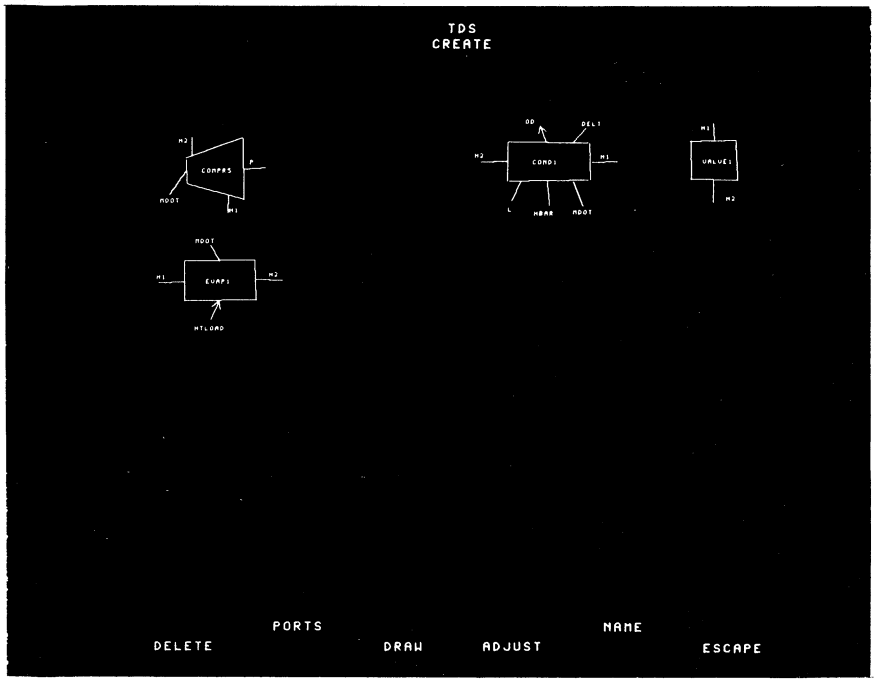


Figure 6.3 The Defined Menu

The formulation of the problem is as follows.

Given:

(a) Copper tube I.D. = .2 inches, O.D. = .25 inches

(b) Heat Load = .3 tons

(c) Refrigerated Space Temperature = $36^{o}F$

(d) Ambient Temperature = $74^{o}F$

(e) The valve is an orifice.

(f) Evaporator Pressure = 24PSI

(g) Compressor Inlet Temperature = $0^{o}F$

(h) Forced Air Convection at both heat exchangers

(i) 36 feet of tubing in each heat exchanger

(j) Compressor Outlet Temperature = $140^{o}F$

(k) Constant mass flow rate = 5 lb/min, F-12

Analysis: (Steady State, negligible kinetic and potential energy changes for flowing Freon-12.)

(a) Compressor

1^{st} Law: $\cancel{Q_i}^{\approx 0} + \dot{m}_i h_i = P + \dot{m}_e h_e$

$$\dot{m}_i = \dot{m}_e$$

$$h_e = h_i - P/\dot{m}$$

(b) Condenser

1^{st} Law: $Q_i + \dot{m}_i h_i = \cancel{P}^{\approx 0} + \dot{m}_e h_e$

$$\dot{m}_i = \dot{m}_e$$

$$Q_i = \bar{h}A\Delta T = \bar{h}L\pi D_o(T_{AVE} - T_{AMB})$$

$$h_e = h_i - \bar{h}L\pi D_o(T_{AVE} - T_{AMB})/\dot{m}$$

(c) Valve

1^{st} Law: $\cancel{Q_i}^{\approx 0} + \dot{m}_i h_i = \cancel{P}^{\approx 0} + \dot{m}_e h_e$

$\dot{m}_i = \dot{m}_e$

$h_e = h_i$

(d) Evaporator

1^{st} Law: $Q_i + \dot{m}_i h_i = \cancel{P}^{\approx 0} + \dot{m}_e h_e$

$\dot{m}_i = \dot{m}_e$

$h_e = h_i + Q_i/\dot{m}$

Algorithms

INSTANCE NAME	PROGRAM NAME	INPUTS	OUTPUTS	ALGORITHM
COMPR	FCOMP	H1,P,MDOT	H2	H2=H1-P*2545/(MDOT*60)
COND	COND1	H1,HBAR,L MDOT,OD,DELT	H2	H2=H1-(HBAR*L*3.14*OD *DELT)/(MDOT*60*12)
VALVE	VALV1	H1	H2	H2=H1
EVAP	EVAP1	H1,HTLOAD, MDOT	H2	H2=H1+HTLOAD/(MDOT*60)

For the computer solution to this problem, see Appendix 6.

6.1.2 Simulation of the Calibration of an Instrumented Cantilever Beam

The idea of this problem is to simulate an instrumented cantilever beam so that a workable instrumentation configuration can be determined for various beams.

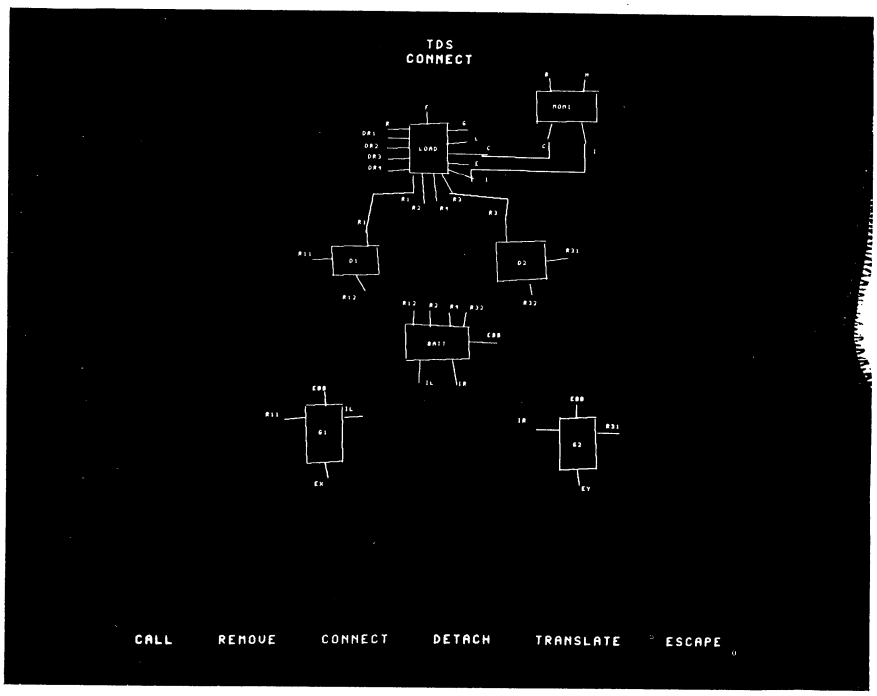
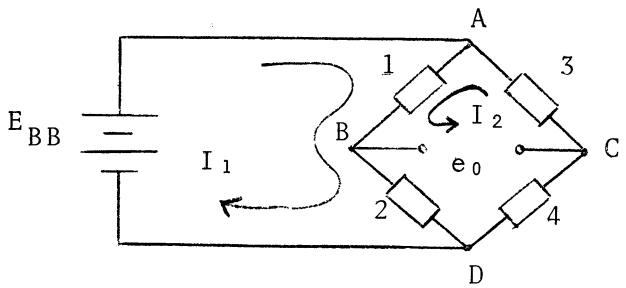


Figure 6.4 A Wheatstone Bridge Simulation

In the above figure, the configuration is not completely connected because the core space in the peripheral computer which is allotted to graphics has been exceeded. This is a good demonstration of the yet-to-be-improved division of labor noted in Appendix 4. Consequently, the solution which follows was done in the $T[TTY] mode of TDS.

The formulation of the problem is as follows.

Conventionally



$$R_{1,4} = R+\Delta R = R(1+\frac{GFLC}{IE}) \qquad G = \frac{\Delta R/R}{\Delta L/L} = \frac{\Delta R/R}{e}$$

$$R_{2,3} = R-\Delta R = R(1-\frac{GFLC}{IE}) \qquad S = \frac{MC}{I} = \frac{FLC}{I} = Ee$$

$$\Delta R = \frac{RGFLC}{IE}$$

$$E_{BB} = I_1(R_1 + R_2)$$

$$E_B - E_A = I_2R_1$$

$$I_1 = I_2$$

$$E_B - E_A = E_{BB}\frac{R_1}{R_1+R_2}$$

$$E_C - E_A = E_{BB}\frac{R_3}{R_3+R_4}$$

$$e_0 = E_B - E_C = E_{BB}[\frac{R_1}{R_1+R_2} - \frac{R_3}{R_3+R_4}] = E_{BB}\frac{\Delta R}{R} = \frac{E_{BB}RGFLC}{IE}$$

Example

For Parameters:

G=2; L=10; B=1.5; E=10,000,000; H=.25; E_{BB}=6;

R=120; F=10

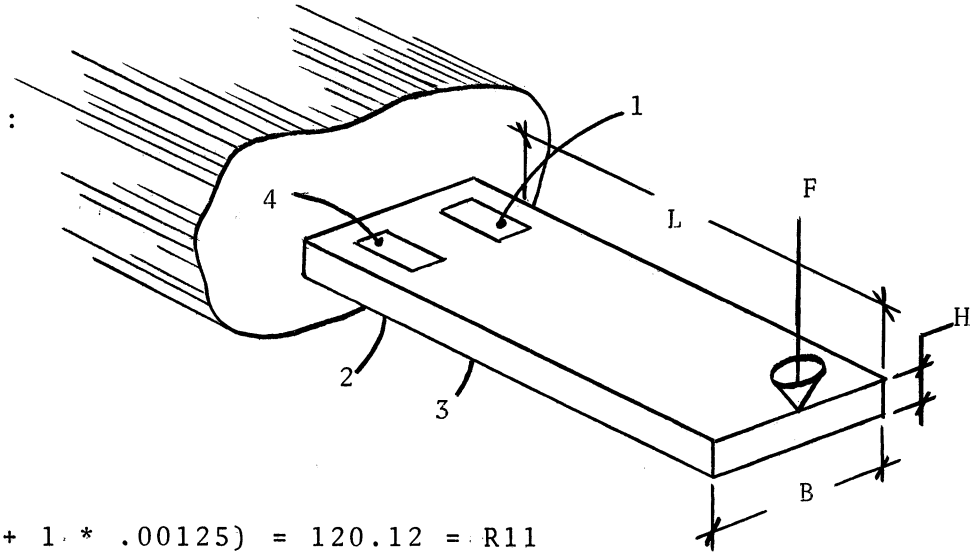
From set up:

DR1 = +1

DR2 = -1

DR3 = -1

DR4 = +1



R1 = 120(1 + 1 * .00125) = 120.12 = R11

R2 = 120(1 +(-1*.00125)) = 119.85

R3 = 120(1 +(-1*.00125)) = 119.85 = R31

R4 = 120(1 + 1 * .00125) = 120.15

IL = 6/(240) = .025

IR = 6/(240) = .025

EX = 6-.025(120.15)

EY = 6-.025(119.85)

EO = EY - EX = .3(.025) = .0075V

Using TDS

1) Eight element functions are required.

   (a) Calculate the strain gage resistance values as a function of gage orientation, materials, geometry and force for a 4 gage bridge on a cantilever. [W1]

   (b) The current draw on the battery of a Wheatstone bridge. [W2]

   (c) Ohm's Law with voltage drop dependent. [W3, W4]

   (d) An oscilloscope whose output is a difference of two inputs. [W5]

   (e) A node whose outputs equal the input. [W6, W7]

   (f) Calculate C & I for a rectangular beam. [W8]

2) See Figure 6.4

   Notes:

   (a) Not only can the applied force be changed, but so can gage mounting and all geometry.

   (b) The solution is <u>not</u> iterative, it is exact for the assumed potential drop across the battery.

Algorithms

INSTANCE NAME	PROGRAM NAME	INPUTS	OUTPUTS	ALGORITHM
LOAD	W1	F,R,G,XL,C,E, XI,DR1,DR2, DR3,DR4	R1,R2, R3,R4	DELTAR=(G*F*L*C)/(E*I) R1=R*(1+DR1*DELTAR) R2=R*(1+DR2*DELTAR) R3=R*(1+DR3*DELTAR) R4=R*(1+DR4*DELTAR)
BATT	W2	EBB,R12,R2, R32,R4	IL,IR	IL=EBB/(R12+R2) IR=EBB/(R32+R4)
G1	W3	IL,EBB,R11	EX	EX=EBB-(IL*R11)
G2	W4	IR,EBB,R31	EY	EY=EBB-(IR*R31)
SCOPE	W5	EX,EY	EO	EO=EX-EY
D1	W6	R1	R11,R12	R11=R1 R12=R1
D2	W7	R3	R31,R32	R31=R3 R32=R3
MOMI	W8	B,H	C,I	C=H/2 I=(B*(H**3))/12

For the computer solution to this problem see Appendix 7.

6.2 A Comparison with Conventional Practice

In the following comparisons "conventional" is taken to mean either paper, pencil and slide rule or interactive teletype. The context will make the meaning clear.

In comparing TDS to conventional methods it becomes clear that an interactive design system forces the designer to determine a quantitative "good enough ... I'm done" criteria. This leads to a saving in time. On the other hand, many of the small tasks that could be performed by a designer on TDS would be too much trouble if not done conventionally. Usually the criteria for enough iterations on a design is that it "seems right."

When considering some "design" systems that do not have their own library or extensive specifyable coaching, both of these capabilities in TDS seem to be valuable assets. A library that can be interrogated brings to the designer's attention existing programs (even in a miniscule library) that would possibly help. Further, having the opportunity to add element functions of personal relevance by merely answering a sequence of queries is a nice feature.

The one feature that is outstanding is the ease with which a problem topology can be manipulated. All the processing related to "bookkeeping" when an element is added, removed or altered is done by the system. This leaves the designer free to consider his "real" problem. In fact, he doesn't hesitate to try some alternatives because they would be too much bother. He can afford to sojourn into unknown areas and thereby possibly

gain some new knowledge. Furthermore, the designer can now afford to handle some traditionally complex problems in a more sophisticated manner. For example, certain "factors" in engineering equations are based on formulas, but they are generally taken to have a particular "value" to avoid the laborious evaluation involved.

6.3 Significance of the Work

The author believes that a system of this type can offer some new opportunities to industry and to the academic community. All of the following comments, however, assume a pool of potential users so that the cost of developing and maintaining such a system can be justified.

6.3.1 For Industry

Industry is constantly forging into new areas whose technology demands more detailed analyses of ever more complex systems. A computer-based design system constructed on the principles outlined, gives the user an advantage while more is being demanded of him. Namely, the effort that he must expend to interact on enough ideas so that he can make a decision is decreased. Further, with the communication at a high level, the designer (1) can affort to "experiment," and (2) does not have to cast aside some alternatives because they would be too much bother to investigate.

Another point, considering that the system is adaptive, is that a mode can be added to yield numerically controlled (N/C) machine tapes directly from drawings. This con-

cept by itself has been developed,[3] but it is yet to be tied elsewhere to a general system design capability such as TDS. In fact, this idea can be carried, using other modes, to the point that manufacturing specifications are generated automatically, possibly even including the best route through available production facilities.

Obviously certain individuals in industry sometimes "design" systems that do not result in manufactured hardware. For example, plant engineers who are concerned with layout, or industrial engineers who are concerned with production and requisition routes and timing do design work. These people would find a ready use for an easy way to communicate flow problems to a central computer for some type of data manipulation.

6.3.2 For Engineering Educators

It seems that the fundamental problem facing engineering educators is to take the third year students with sound technical fundamentals and mold them into engineers. These students must make the transition from textbooks to actual engineering (design) problems. The contribution that a system such as TDS would make is that during a short university term, the fledgling engineer would get an opportunity to make many essentially costless mistakes with the simultaneous advantage of his instructor's commentary. The use of TDS in an academic environment seems to be natural because the analysis programs developed in the terminal semesters would feed the design system while certain term projects could extend it.

3 Marvin T. Ling, The Logical and Analytical Structure of the Computer-Aided Design Process as Applied to a Class of Mechanical Design Problems (University of Michigan, 1964).

CHAPTER 7

CONCLUSIONS AND OPEN PROBLEMS

This study has concentrated on increasing the effectiveness of the engineering designer by providing him with an interface to a new computer-based environment. The purpose of the interface is to allow him to consider more problem component interconnections quantitatively, thereby getting more data per unit time on which to base design decisions. As conceived and constructed, the prototype interface is a logical peripheral device remote from a large central computer. The performance of the prototype design system seems to be sensitive to a number of distinct but related parameters: drawing speed, graphic data structure operator speed, peripheral task switching speed, central file switching speed, data retrieval speed, time to increase allotted storage and especially the other simultaneous users on the central computer.

7.1 Conclusions

From the operation of This Design System, it seems obvious that manipulating the topology of a design problem is best done graphically. TDS not only allows the designer to manipulate information readily (unbind and re-bind previous decisions), but having a constantly up-dated picture of the problem topology as a whole seems to give him quite a "feel" for the problem being considered. Furthermore, it is very easy in the light-pen mode to manipulate a problem topology and hence a problem solution. In this implementation,

topology specification can be done either graphically or via the teletype. With this device independence, a designer can determine for a particular problem what will be most effective for him.

The notion of giving the designer the right to a priority interrupt at any time while he is using the system has proved to be invaluable. It seems that no matter how familiar a designer becomes with the system, he seems to make conceptual errors that are relevant to the problem context and hence inevitably require some back-tracking. In practice, once the need for back-tracking is realized, the designer wants to do it immediately. The priority interrupt allows him to do so, and does not distract his train of thought.

The idea of miscellaneous program coupling has proved to be valid and valuable. The same person, writing in the same language, at different times, on the same operating system, tends to format his programs differently; so that, even for this case, which would seem to be trivial, if there is a mismatch at all, the system would not work unless automatic program coupling was in effect.

The human factors of file switching as implemented seem to be very poor. The reason is that while file orientation for the system is required in this implementation to obtain the operation desired, the actual speed involved is so slow as to be distracting. So, while the concept of a file-oriented central computer system is desired, it seems that there has to be an entirely different concept for carry-

ing this out, and somehow that has to be performed at a much higher rate of speed. The probable solution is to provide for a high volume of traffic for file switching.

The division of labor, as outlined in Appendix 4, seems to be quite good. It is without a doubt correct to have the interpretable graphic data structure in the peripheral computer. The human factors of drawing seem to require that in that particular mode, the response be almost instantaneous. However, because of the limited size of the peripheral computer, one correction that should be made to the division of labor is that a concatenated version of the display file should be kept locally, and the expanded version in the central computer.

The idea of keeping all the design data in an associative structure has been demonstrated, within the development time of the system, to be a good principle. The reason is: a certain portion of the system was first built on a list structure type basis, and when changed to TRAMP[1] became much smaller and faster.

The entire concept of the library as conceived and implemented seems to be excellent in that it gives good access to varying levels of library detail. However, there must be a better way to store not only the information relevant to a library routine, but the actual routine contents. Unit conversion, while it is not very difficult to conceive and develop, is very laborious to implement. In order to be retrieved

---

[1] W.L. Ash and E.H. Sibley, TRAMP: A Relational Memory with an Associative Base (Ann Arbor, The University of Michigan, June 1968).

in the future, these data must all be entered. It would be a good future extension of the work to have "key" units readily retrievable in all their existing combinations so that a designer could quickly determine if his needed units are stored in some form. (e.g., the designer might say BTU, and receive BTU/HR, BTU/HR-FTSQ-F).

The Build mode, or the mode in which the Vari-port link operates as a modulator, seems to work extremely well. It is a very versatile mode, and, in fact, for the specification of reticulated systems, will tolerate almost any user inconsistancy and handle most any user error.

One of the most illuminating ideas to result from this work is the notion of "the NEW BUILD." Following the development of the proof for Theorem 4.3, and the successful implementation of the system, it became evident that a more general system probably could be devised. Using the set-theoretic definitions developed here, it should be possible, using macro-operators such as "intersection" etc., to topologically order a user-specified cyclic network. This would reveal to the designer <u>where the trial values should be</u>. In other words, this algorithm would return to the designer those points at which he should estimate the values of the flows between entities. The implication of this is great when one considers optimization, where an increase in the number of variables to be handled is accompanied by more than a linear increase in the effort required to optimize.

Optimization of systems has not been approached in this study. Obviously some serious stability problems could occur as the system variables are related to different element functions.

Finally, the development and implementation of a system based on these principles must be economically justified. Now that this prototype system has been built, new implementations will of course follow more inexpensively. The ultimate test however will be to have the next generation TDS available to both industrial and academic users so that effectiveness data can be collected.

7.2 Open Problems

In light of these foregoing conclusions, there are many problems which would be worthwhile to investigate.

1) How should the execute mode be built so that all use of formatting is transparent to the user?
2) What should the construction of the central computer portion of This Design System be so that other type data structures can be user-requested?
3) What should be the construction of the NEWMODE algorithm so that it has the capacity to redo or rewrite the executive in the peripheral computer?
4) When one talks of a high-level language for the designer, what can be said about "high-level" quantitatively? Furthermore, what is the contribution of format to this level, and what is the contribution of speed to this level?

5) What sort of experiments could be devised to test the percent improvement achieved by future alterations of a system such as TDS?

6) What are the relative bit-rates for information transmittal from the user to the CPU for graphics versus the teletype?

7) What qualitative way can be used to tell a priori whether this system will be "good" for a particular design problem or not?

8) The adaptive characteristic already built into TDS allows new modes to be added to the newly created data linkages but obviously does not fill them in. Consequently, what type language should be developed for a user to employ in "filling in" a newly added data linkage? Should this preferably be graphic?

9) Given that many engineering systems are dynamic in nature, what type data structure and what operators are required for direct formulation from graphics to solve differential equations?

10) Given that in this system as presented, graphics does not connote data, what type data structure is required such that, for example, the sketching of blade shapes to connote data, the sketching of orifices, etc., will be allowed within the reticulated system environment?

11) How does the concept of the reliability of engineering entities enter into the design process, and how is it implemented in a design system?

12) Similarly, how are manufacturing considerations integrated into a design system?

13) What effect on both the data structure and method of operation does hybrid data acquisition have?

14) Given that there are some applications, for example the design of integrated circuits which require multi-level graphics (i.e., "stacked" two-dimensional pictures that can appear one at a time), what type data structure is required for an i^{th} level subset of what is described herein as the highest active level?

15) What characteristics of a design system are required such that illegal connections based on context interpretation are disallowed?

16) What is required in the way of automatic documentation such that an abstract is changed when the designer changes the algorithm?

17) How can a library I/O routine interpret its context and talk meaningfully about things that it might or should contain, for example, an implementation of an ELIZA-type program?

18) How can the library realize the hit and retrieval rate on its contents and thereby change the way in which it retrieves relatively irrelevant material? And, based on some algorithm, how can the library suggest to the user the possible deletion of certain totally irrelevant material?

19) Where does one begin with trial values to most effectively "home-in" on the solution? (The NEW BUILD.)

20) What is required to have graphically inputted parameter values?

21) What algorithms can a reticulated network be subjected to that will order the P-list based on a minimum number of trial values, thereby dictating where the designer should approach his problem to get a solution based on the fewest estimates?

22) What does fewest trial values mean to optimization?

APPENDIX 1

THE USER'S GUIDE TO TDS

Figure A1.1 Man-Computer Configuration

The idea of having the teaching machine characteristics in TDS is so that you can learn about the system, and ask questions about your own work if and when the need arises. Examples of this are shown in Figure A1.2 and A1.3. However, before attempting to use TDS, you should understand its macro capabilities. An outline of these capabilities follows. Not included in the following are the constantly changing details associated with physically loading and starting this developmental system.

A1.1 The Basic Modes

When you are asked to "TYPE DESIRED MODE NAME, OR

```
 06-05-68
SIGNED ØN AT: 13:05.02


HELLØ, YØU ARE NØW USING TDS.  DØ YØU NEED
INSTRUCTIØNS FØR INITIALIZING THIS DESIGN SYSTEM?
PLEASE TYPE "YES" ØR "NØ".

YES
FINE, ARE YØU CALLING FRØM A "TELETYPE", A "338"
ØR A "2250"?

TTY
Ø.K.
YØU ARE NØW READY TØ USE TDS UNDER TELETYPE CØNTRØL


WØULD YØU LIKE INSTRUCTIØNS FØR USING TDS, AND A
SUMMARY ØF ITS CAPABILITIES? "YES" ØR "NØ"

NØ
Ø.K.



TYPE DESIRED MØDE NAME, ØR INSERT
MØRE INFØRMATIØN INTØ PRESENT MØDE.

$T


 THIS IS T/I


 IF YØU WØULD LIKE INSTRUCTIØNS
FØR USING THIS MØDE PLEASE TYPE
"YES" ØTHERWISE "NØ".
```

Figure A1.2 Sample TDS Signon

```
$BUILD
  SAVE TASKS WHEN LEAVING SUPPLY-T/I.

      IF YOU NEED INSTRUCTIONS ON SELECTING
      THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
      TYPE YES' OTHERWISE NO'.


 THE PRESENT INSTANCE IS " COMPR5 "


NO

      PLEASE MAKE YOUR SELECTION NOW.


=FCOMP

      IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
      MATCH THE PORTS OF THE INSTANCE "COMPR5"
      WITH THE PORTS FOR SUBROUTINE "FCOMP
      PLEASE TYPE YES' OTHERWISE NO'.

NO

      PLEASE USE THE FOLLOWING
      TO MAKE YOUR MATCHES.


INSTANCE INPUTS: MDOT  ;H1    ;HPWR   /INSTANCE OUTPUTS: H2

SUBROUTINE INPUTS: 056H1;056P;056MDOT /SUBROUTINE OUTPUTS: 056H2

H1=056H1
" INPUT MATCH NO. 1"


MDOT=056MDOT
" INPUT MATCH NO. 2"


HPWR=056P
" INPUT MATCH NO. 3"


H2=056H2
" OUTPUT MATCH NO. 1"
```

Figure A1.3 Sample TDS Coaching Comments

INSERT MORE INFORMATION INTO PRESENT MODE," as in the last lines of Figure A1.2, you should type one of the following mode commands. Instructions will follow.

The basic modes of operation (explanations begin below) are:

1) $T
2) $BUILD
3) $EXECUTE
4) $LIBRARY
5) $NEWMODE

These modes are all independent, meaning that you can enter any of the modes at will. You are not faced with a limited, order-dictated (sequential), system. Be sure that you understand when you get to section A1.3 that those are explicit commands, not modes. They have arguments!

A1.1.1 "$T"

Among all the modes of operation, this mode is unique because it allows you to enter the topologies of your problems either through the teletype or via the graphic interface. The proper linkages are made for you whenever you answer the second question in Figure A1.2.

If you state that you are using TDS via a teletype, then the T/I mode appears to be quite different from a "338" sign on. Instead of hitting light buttons and typing six character names to enter instances and ports, a group of operators (see below) are defined which perform the same net function as light buttons in the "338" display. All operators

are recognized by a "%" and then the first character of the actual operator name. Available operators are (see example in Appendix 7 for a typical implementation):

%INSTANCE/NAME/

%PORT/NAME//TYPE(i.e.,P,T,I,O):PORT NAME/

%DELETE/NAME//PORT NAME/

%CALL/NAME/

%REMOVE/NAME/

%MAKE A CONNECTION/NAME:PORT NAME/
/NAME:PORT NAME//NUMBER/

%BREAK/NUMBER/

The operators have arguments, each one of which is contained within two slashes "/".

The possible arguments are:

NAME - instance name

PORT NAME - port name

TYPE - type of port (output, input, parameter, trial value)

NUMBER - a unique number (which you arbitrarily think of) for each connection

A definition of each operator follows:

%I/NAME/ - is the naming procedure (computer prints "instance "name" is now in the menu")

%P/NAME//TYPE:PORT NAME/ - enables you to put ports on an instance <u>previously</u> named by "%I". More than one port name may be entered at once using

";" as the delimiter. However, each must have its own "type" specification.

%D/NAME//PORT NAME/ - allows you to delete ports which have been attached to instances. (no concatenation allowed).

%C/NAME/ - calls the instance specified out of the menu and into the active program.

%R/NAME/ - allows you to remove a program from the active program and put it back in the menu.

%M/NAME:PORT NAME//NAME:PORT NAME//NUMBER/ - makes a connection between input and output ports on instances in the active program. The number allows you to easily break such a connection later, if desired.

%B/NUMBER/ - allows you to break connections made with the "%M" operator merely by restating the number originally stated in the "%M".

Concatenation may be used in only the "%P" operator. In all others only a single port, connection, or name may be acted upon in any one statement.

If you state that you are using TDS via a 338, then you will enter the topologies of your problems graphically. To do this you need to know how to use the first light buttons that appear on the screen (see Figure A1.4) when this mode is declared.

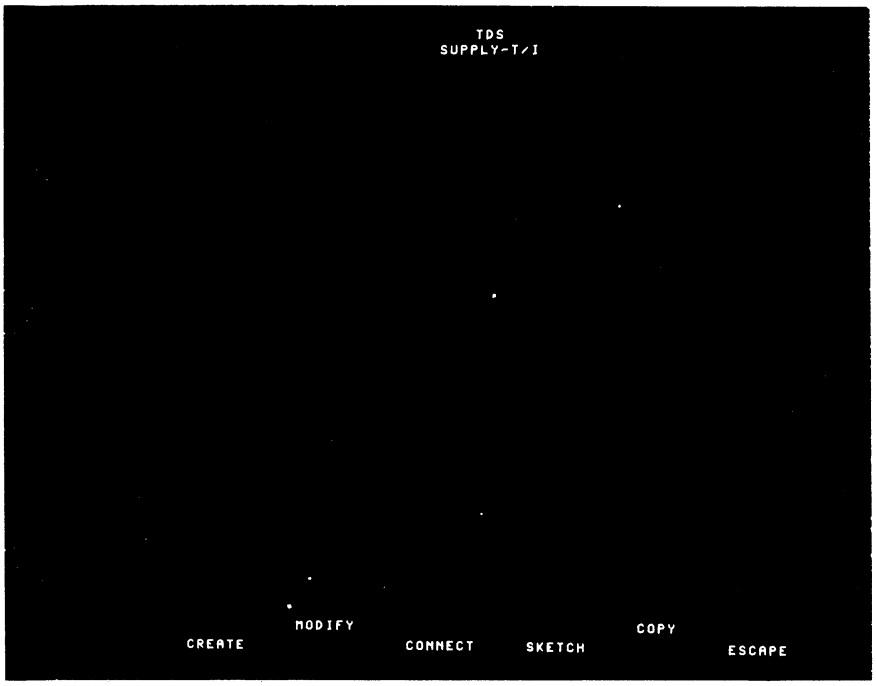


Figure A1.4 "SUPPLY-T/I" Picture Frame

The functions of the light buttons shown in Figure A1.4 are as follows. (To activate a light button, point at it with the light pen and open and close the shutter.)

1) CREATE - switches to the "CREATE" frame.
2) MODIFY - brings up the menu of previously defined symbols (if any), and puts the one which you point to with the light pen on the screen, along with the "CREATE" frame.
3) CONNECT - switches to the "CONNECT" frame.
4) SKETCH - switches to the "SKETCH" frame.
5) COPY - presently inoperative. The purpose will be to allow multiple copies (but with unique names) of the same instance.

6) ESCAPE - removes all frames and has the teletype request another mode.

The "CREATE" frame is shown in Figure A1.5. Using this frame you will define your graphic instances as shown in Figure A1.6.

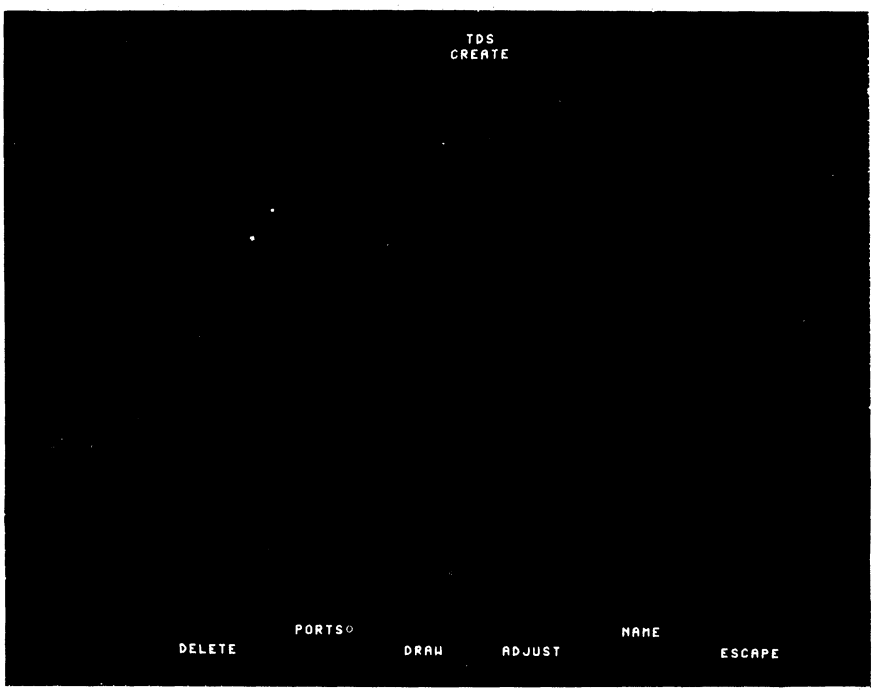


Figure A1.5 "CREATE" Picture Frame

The functions of the light buttons shown in Figure A1.5 are as follows.

1) DELETE - allows you to delete any user-defined labels or vectors on the screen by pointing at them with the light pen.

2) PORTS - allows you to draw ports on graphic instances. When each port is complete the frame switches to the

"LABELS" frame and you can then name the port. Drawing and naming are described below.

3) DRAW - this button provides you with a tracking cross that you drop (by closing the shutter) on the screen where you want the line to begin. You then pick up the "speck" which appeared when you dropped the cross and "stretch" it as you please. Where ever you close the shutter (i.e., drop the tracking cross) will be the terminus of the line.

4) ADJUST - in the event that you want to move a line, you can move one end at a time using this button. The end of the line you point to will follow the light

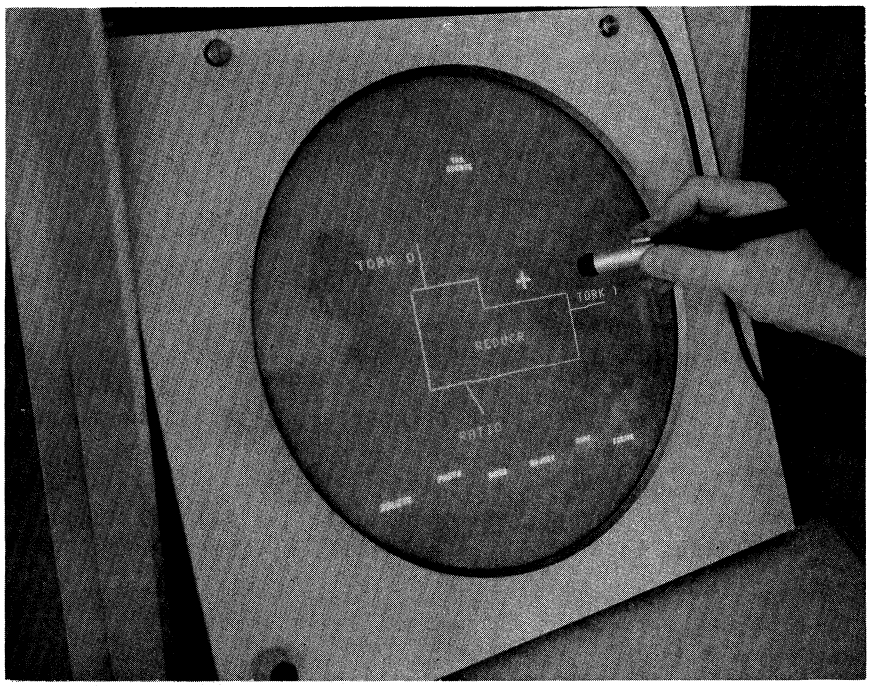


Figure A1.6 Defining a Graphic Instance

pen as in drawing.

5) NAME - after "hitting" this button you will get a tracking cross which you drop where you want the center of the six character name to appear. When you drop the cross, the teletype will request a six character name (see Appendix 6 for an example).

6) ESCAPE - this button puts the current instance in the menu and switches to the "SUPPLY-T/I" frame.

The "LABELS" frame is shown in Figure A1.7. This frame is only called by the "PORTS" light button in Figure A1.5. From all four light buttons you get a tracking cross as for "NAME" and drop it where you want the teletype-requested port label to appear.

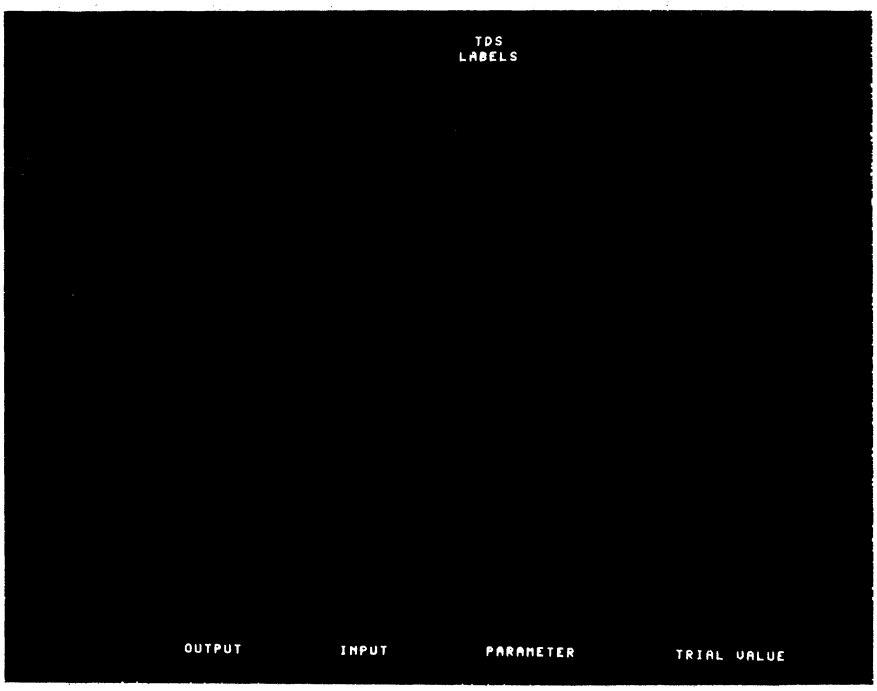


Figure A1.7 "LABELS" Picture Frame

The "CONNECT" frame is shown in Figure A1.8. The functions of the light buttons are as follows.

1) CALL - brings up the menu so that you can select (as you do in MODIFY) another instance that you have defined and have it inserted in the problem topology.
2) REMOVE - the inverse of CALL.
3) CONNECT - lets you connect ports of the instance you have "CALLED" into the topology.
4) DETACH - the inverse of CONNECT.
5) TRANSLATE - lets you move instances around (shuffle them) in the topology.
6) ESCAPE - takes you back to the "SUPPLY-T/I" frame.

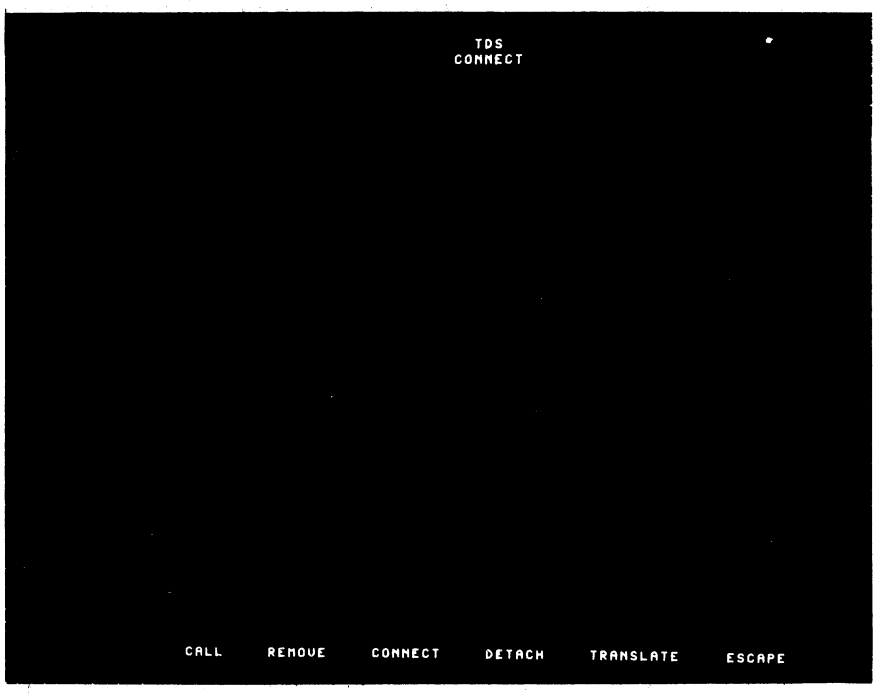


Figure A1.8 "CONNECT" Picture Frame

The "SKETCH" frame shown in Figure A1.9 has the buttons DELETE, DRAW and ADJUST which work the same way as the buttons of the same name in Figure A1.5, except that no data are sent to the central computer; this mode is strictly to give you a "scratchpad." The ESCAPE button switches you to the "SUPPLY-T/I" frame.

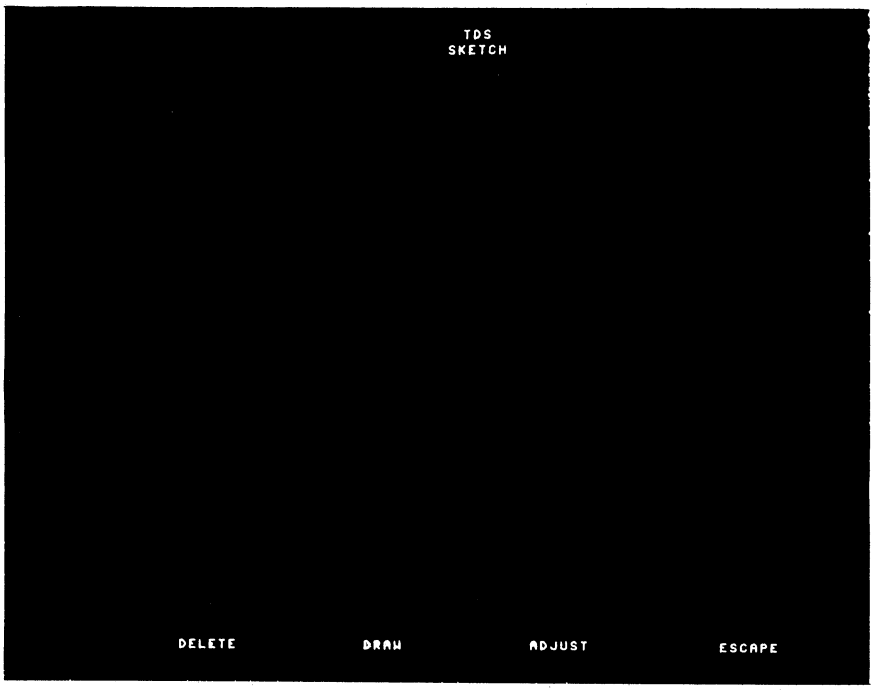


Figure A1.9 "SKETCH" Picture Frame

A1.1.2 "$BUILD"

After having passed through the "$T" mode, you can enter the "$BUILD" mode. When the teletype says "Please type desired mode," you type $BUILD and the teletype will acknowledge by saying "This is BUILD." From that point on, the comments which come out on the teletype should be sufficient to coach you through the use of the BUILD mode. The purpose

of the BUILD mode is two-fold. First, the ports specified on all the instances must have values and engineering units associated with them. BUILD will ask explicit questions as shown in Figure A1.3 about a particular instance, and its particular ports. The other function of BUILD mode is to determine which analysis program previously stored in the library is to be considered as corresponding to that particular graphic instance.

A1.1.3 "$EXECUTE"

After passing through the BUILD mode, you can type "$EXECUTE" and be admitted to the EXECUTE mode. A confirmation is required so that the central computer will not be asked to execute a null program. Between the execution of subsequent analysis programs, the user has the option, if he so desires, to monitor the input and output of each of the instances as the network is analyzed. In every case, the information in the library is sufficient to allow automatic re-formatting of analysis program inputs and outputs. Hence, there is nothing that you have to do to couple programs more than merely specify what program goes with what instance.

Although the above material seems to indicate a $T-$BUILD-$EXECUTE order, you may immediately type $EXECUTE upon entering the system. If no prior information exists which can be operated on, you will be cajoled into requesting another "available" mode.

A1.1.4 "$LIBRARY"

This mode can be entered at any time. It too is completely independent of all other modes. Within this mode

are several sub-modes listed below.

%ESTABLISH

The purpose of this sub-mode of the library is to enter a subroutine or program into the memory banks of the library. All pertinent information related to this particular program are entered at this time, including format, units, and language. The source and/or object programs are loaded into storage by merely answering the questions asked by the teletype. A permanent record of input and output units and formatting is automatically made available by the sub-mode before loading the source and/or object programs.

%DESTROY

If a program which is documented in the library is no longer desired, this mode will completely remove it from library storage. It not only removes every location where information had been about how to reach this subroutine, but it also removes the object and the source decks. In order to use this, just follow the questions posed by the teletype.

%ALTER

This allows you to alter programs already in the library, for example, change the number of ports, etc. However, the program must be reloaded into the library and information concerning it changed. This mode automatically "lists" the program to be altered in the central computer and then reloads it exactly where it was.

A1.1.5 "$NEWMODE"

This mode gives you the opportunity of adding another

capacity such as NEWMODE or BUILD or LIBRARY to TDS. All that you need to do is merely to state $NEWMODE and answer the subsequent questions. From that point on, when the teletype says that your NEWMODE has been entered, you can then say $[whatever the name of your new mode happens to be] and you will have linkages to that routine. The contents of that new mode are to be provided by you.

A1.2 The Command Language Interpreter

Certain special symbols, namely "$" and "&" are recognized as prefixes to commands in the central computer portion of TDS. The & is used automatically by the interface, and never explicitly by the designer. The commands available are as follows.

A1.2.1 "$CORRECT"

This provides the capability of correcting your mistakes. For example, if you name an instance John instead of Jack, you need only give this command and then answer the questions with respect to what you wish to correct.

A1.2.2 "$QUESTION"

This provides the capability of questioning TDS concerning what is in the library and what has been produced or entered into any mode.

A1.2.3 "$ALTER"

This mode gives the ability to change parameters of trial values for the optimization of programs specified in TDS. In the future it will be possible to write an optimization program which itself calls on this command.

A1.2.4 "$TRANSFER"

This allows you to move from mode to mode but always return to the mode that you just left. Actually, this mode serves as a bypass.

A1.2.5 "$ENTRY"

This is a null command which produces a carriage return and two line feeds. Any comments on the same line following the command will be printed out as "your inserted comments" without being acted upon.

A1.2.6 "$CREATE"

This mode allows you to stop whatever you are doing and leave TDS. In order to return you type $SOURCE ENTRY instead of $SOURCE TDS.

A1.2.7 "$BYE"

When you finish completely with a program in TDS, you need only type "$BYE" and everything that you have done will be removed from memory and TDS will be completely reinitialized with all virtual storage removed in the central computer. You are then signed off the central computer automatically.

A1.3 Explicit Commands

In addition to the above "mode commands," there are explicit commands which involve parameters and give immediate information to the user. These are NOT modes.

A1.3.1 "$CORRECT/Old Association//New "V" part/

Since data is stored by TDS in an associative structure, the following is the form for any association:

$$[A\ (\emptyset) = V]^{1}$$

Where A denotes attributes such as "value" or "units"

$\emptyset$ denotes objects such as port or subroutine "name"

and V denotes values such as "true value" or "units"

The parts of the association are delimited by a ";". For example, a typical $CORRECT command would be"

$CORRECT/A;Ø;V1//V2/

Note: This command allows the user to change only the "value" portion of the association. As soon as the correction is made the new addition to the memory will be printed in the form A(Ø) = V. The old association must exist. If it does not, complaint will be made.

A1.3.2 $QUESTION/A;Ø;V/

This is merely a direct question on the memory. There are several available forms for this command. In its full form (above) the question seeks a "yes," "no" or "maybe" (i.e., multiply defined) answer. (It is actually asking—does this association exist in the memory?) To extract more information from the memory than just "yes," "no" or "maybe," the user can leave any one or two of the locations available empty. Several examples will serve to clarify this point.

---

[1] J.A. Feldman, "Aspects of Associative Processing," (Cambridge, Mass., Lincoln Laboratory, April 1965).

1) $QUESTION/;Ø;/

This says: Give me all the attributes and values of the object "Ø."

2) $QUESTION/A;;V/

This says: Give me all the objects which have the attribute "A" and the value "V."

The final form is abbreviated and provides faster answers concerning the "VALUE" component of the association. The form is:

$QUESTION/A;Ø/

Typical examples are:

1) $QUE/A;Ø/

This says: Give me all values of object "Ø" which have attribute "A." This is exactly the same as "$QUE/A;Ø;/"

If all three positions are eliminated, i.e., $QUE/;;/, an associative dump of the entire TRAMP[2] memory will be made.

Warning: This can be enormously long and is enormously expensive.

A1.3.3 $ALTER/PORTNAME:Old Value;New Value/

or

$ALTER/PORTNAME:New Value/

Both of the above forms of the "$ALT" command have the same function: to alter the value of some parameter or trial value in the system. The "PORTNAME" is the subroutine

---

[2] W.L. Ash and E.H. Sibley, TRAMP: A Relational Memory with an Associative Base (University of Michigan, Concomp Technical Report, June 1968).

input which is to have its value changed. The "Old Value" is the value the port presently has. The "New Value" is the value the user wishes it to have now.[3] If the first form is used, a check will be made to see if the value stated is accurate. If it is not correct a complaint will be made.

If the second form is used, no check is made, and the new value is entered into the memory. In either case, acknowledgment of action taken by the system will appear.

A1.3.4 $CREATE/[a number][a space][a file name]

This explicit command is much the same as the mode call except that it allows the user to specify where he wishes everything stored, and what line number this program is to have.

If "file name" is missing, the default for reinitializing is "entry," and for memory storage is "memory."

If "a number" (line number of program) is missing, the number will be automatically set to "1."

To reenter TDS by reinitializing where you left off, you need only give the command "$SOURCE" "file name" "[number]" where file name and number are those explicitly or implicitly stated in the $CRE command.

Note: if "file name" is specified, the memory is stored in line "1000+number" of that file.

A1.3.5 $DEFINE/CONVERT;[this]:[to this];[x by this];[+this]/

This command allows you to define a mathematical conversion. An example is:

$DEF/CONVERT;IN:FT;12:0/

---

[3] See Appendix 6 for an example of this command.

A1.3.6 $TIME, $TIMER ON, $TIMER OFF

This explicit command allows the user to do one of two things: (1) he can obtain the exact time in the following form: Hours:Minutes (in 24 hour time), or (2) he can obtain the elapsed time between a "$TIMER ON" command and a "$TIMER OFF" command. Elapsed time is given in minutes and fractional parts thereof.

A1.4 Central Computer Command Analyzer

If two dollar signs begin an input line, the line is recognized as a central computer command. Control is automatically given up to TDS and the command is acknowledged by the central computer. Immediately thereafter (unless it was a signoff command), virtual memory is cleared and you are given control. For example:

```
$$SIG
```

This would sign you off of the central computer as soon as you are signed off of TDS.

A1.5 User Specifiable "Attributes"

The following are legal attributes to be used in the "A" position of $ALTER, $CORRECT, $DEFINE or $QUESTION.

FORMS - possible forms of a unit's format

UNITS - units for a port

SIPORTS - subroutine input ports

SOPORTS - subroutine output ports

CONVERT - unit conversions

VALUE - port values

EXTRA - all non-connected SOPORTS (defined only after one pass through Execute)

APPENDIX 2

HISTORY OF COMPUTER GRAPHICS

The following history of computer graphics is drawn largely from an article entitled, "Progress in the Computer Field," L.C. Hobbs, editor, Computer Group News of the IEEE, Vol. 1, No. 7, July 1967.

Graphic output has been a part of computer technology almost since its inception. Whirlwind I contained an on-line display and camera recorder in 1951. As commercial computing drifted into batch processing in the late 1950's, off-line graphic plotters were developed in the form of large universal drafting machines (ink plotters) and film recorders. Plotters are still unparalleled for large, high-resolution graphic presentation. Film recorders, however, are much faster but have limited size and resolution. Where adequate software has been provided, these graphic devices have become important adjuncts to the line printer for computer output. In the intervening years, both these forms of graphic output have assumed more important roles. In addition, several novel uses of film recorders to produce movies have demonstrated the versatility of this form of computer output in communicating to man.

More dramatic development has occurred in the area of on-line computer graphics, that is, displays. For years, displays were the foundling child of the military, although research work in the field was being performed earlier at MIT, IBM, SDC, General Motors, and similar institutions; not until 1962 was real interest aroused for on-line displays as a tool

to aid the problem solver. That year, Culler and Fried demonstrated their system for aiding in mathematical analysis. Then, in 1963, Sutherland's SKETCHPAD program illustrated the power and flexibility of the on-line computer and graphic display as an input-output medium. These programs awakened the scientific community, but both were done on equipment that was not designed for this type of work. It was still terribly expensive to be on-line to a computer of any significant size. Then, in 1964, time-sharing of large computers was demonstrated, giving promise of moderate cost for on-line operation of a powerful computing complex. Suddenly displays were being considered by large groups of computer users. "Graphics" joined the list of popular words.

As the interest in displays for scientific computing rose sharply, equipment manufacturers began offering displays tailored to this new market that were less costly and easier to program than their military counterparts. Several companies offered new displays, such as the RAND tablet, and new graphic input devices were developed and demonstrated.

As these pieces of equipment were developed and people began writing programs to use them, reports from earlier pilgrims in the field were being published. Particularly impressive were General Motors' efforts in using computer displays to aid in drafting and design.

In 1966 applications programs using on-line displays began to appear. These spanned a wide range of fields in science and engineering. The aircraft and aerospace industries

with promise of large pay-offs pursued computer-aided engineering design and analysis. Mathematical analysis programs received quite a bit of attention. Circuit design and layout was also quite popular. More esoteric uses were reported, for instance, molecular model building, automatic programming, and even textile designs.

As displays were put into more use, the need for unifying graphic languages and facilities for manipulating data structures dynamically became more evident. For this reason, software developments such as AED, CORAL, PL-1, GPAK, etc., significantly influenced graphics application programs. In addition to military operating systems, General Motors Research, Lockheed Georgia, Bell Telephone Laboratories, Boeing, Ford, and other concerns are developing operating systems that are somewhat generalized for their own purposes.

If a trend in the field of display hardware is to be noted, it is toward two distinct classes of equipment. The first is an inexpensive remote terminal and the second a sophisticated, dynamic, expensive console. These remote units are aimed at providing low-cost consoles that communicate over standard telephone lines. The sophisticated consoles are being designed with versatile performance as their key feature. The power of the dynamic display with real-time interaction has been recognized, and to provide this, small general-purpose computers are being intimately coupled with displays (sharing memory). These units can then talk to a large time-sharing system, or stand alone. Features such as picture

subroutines, real-time rotation, translation, and scaling, light pen tracking, etc., are being provided, sometimes with hardware and sometimes with software. Hardware is being organized to make these functions easy to program and to require a minimal load for the computer.

The very latest display consoles will have hard-wired micro programs in them such that routine display support will be available essentially as macro instructions.

APPENDIX 3

DETAILS OF TDS' IMPLEMENTATION

The programming required to produce TDS can logically be divided into four sections: the peripheral and central operating system, and the "user" routines at each location that are uniquely TDS.

The peripheral executive, the SEL (Systems Engineering Laboratory) Executive System, will be described in detail in a forthcoming report.[1] Its implementation is currently being completed for a DEC-339 computer. The original executive algorithms and much of their coding are due to Mr. Jackson. Mr. Jackson and the author modified and developed the system to its present state for the DEC-338 computer employed in this investigation. In light of the forthcoming report, the extent of the explanation for this executive will be confined to a description of the operators available.

The central executive is the Michigan Terminal System (MTS)[2], and the language used is TRAMP.[3] The reader is referred to the indicated references for the operating details of both the time-share system and the associative-base language.

The peripheral and central sections of TDS are macro

---

1 A Systems Engineering Laboratory Report on the SEL Executive, by James H. Jackson, should be released in the summer of 1968.

2 MTS: Michigan Terminal System (University of Michigan Computing Center, Ann Arbor, 1968).

3 W.L. Ash and E.H. Sibley, TRAMP: A Relational Memory with an Associative Base (Concomp Technical Report, University of Michigan, June 1968).

flow-charted to make clear how the functions of TDS are obtained. Listings would be out of place here, but are available to qualified parties from the Concomp Project office.[4]

A3.1 The SEL/338 Executive System

The most concise way to describe the operators available in this executive is to list the system transfer vector as in the following table.

SYSTEM FUNCTIONS

```
XQC      DC    QC       CLEAR QUEUE
XQIN     DC    QIN      ADD ITEM TO QUEUE (F)
XQOT     DC    QOT      FETCH ITEM FROM QUEUE (F)
XTSK     DC    TSK      SCHEDULE A TASK
XTSKN    DC    TSKN     START NEW TASK
XTA      DC    TA       ALLOCATE I/O UNDER MASK
XTR      DC    TR       RELEASE I/O UNDER MASK
XKBC     DC    KBC      CLEAR KEYBOARD BUFFER
XKBS     DC    KBS      GET CHARACTER FROM KEYBOARD
XKBSE    DC    KBSE     CALL KBS AND ECHO
XTPC     DC    TPC      CLEAR TELEPRINTER BUFFER
XTPS     DC    TPS      SEND CHARACTER TO TELEPRINTER
XTRTDS   DC    TRTDS    BUILD A CHARACTER INTO A DISPLAY FILE
XDW      DC    DW       WAIT FOR DISPLAY TO COMPLETE FRAME
XDE      DC    DE       ENABLE DISPLAY INTERRUPTS (GLOBAL)
XDD      DC    DD       DISABLE DISPLAY INTERRUPTS (GLOBAL)
XDP      DC    DP       SET LP TASK,AC=ADDR OF TASK,0 IF NULL TASK
XDA      DC    DA       READ DISPLAY ADDR,LINK=0,BANK=1,=1,BANK 2
XDY      DC    DY       READ Y COORD
XDX      DC    DX       READ X COORD
XDO      DC    DO       READ OWNER LEVEL (F)
XDSPBS   DC    DSPBS    SET PB TASK
XTKST    DC    TKST     START TRACKING AT GIVEN COORDINATES
XTKON    DC    TKON     TURN TRACKING ON
XTKOFF   DC    TKOFF    TURN TRACKING OFF
XTKSKP   DC    TKSKP    SKIP IF TRACKING IS OFF (F)
XTKGX    DC    TKGX     READ X TRACKING COORDINATE
XTKGY    DC    TKGY     READ Y TRACKING COORDINATE
XTPB     DC    TPB      FEED TELEPRINTER BUFFER
```

---

[4] Concomp Project, The University of Michigan, 611 Church Street, Ann Arbor, Michigan 48104.

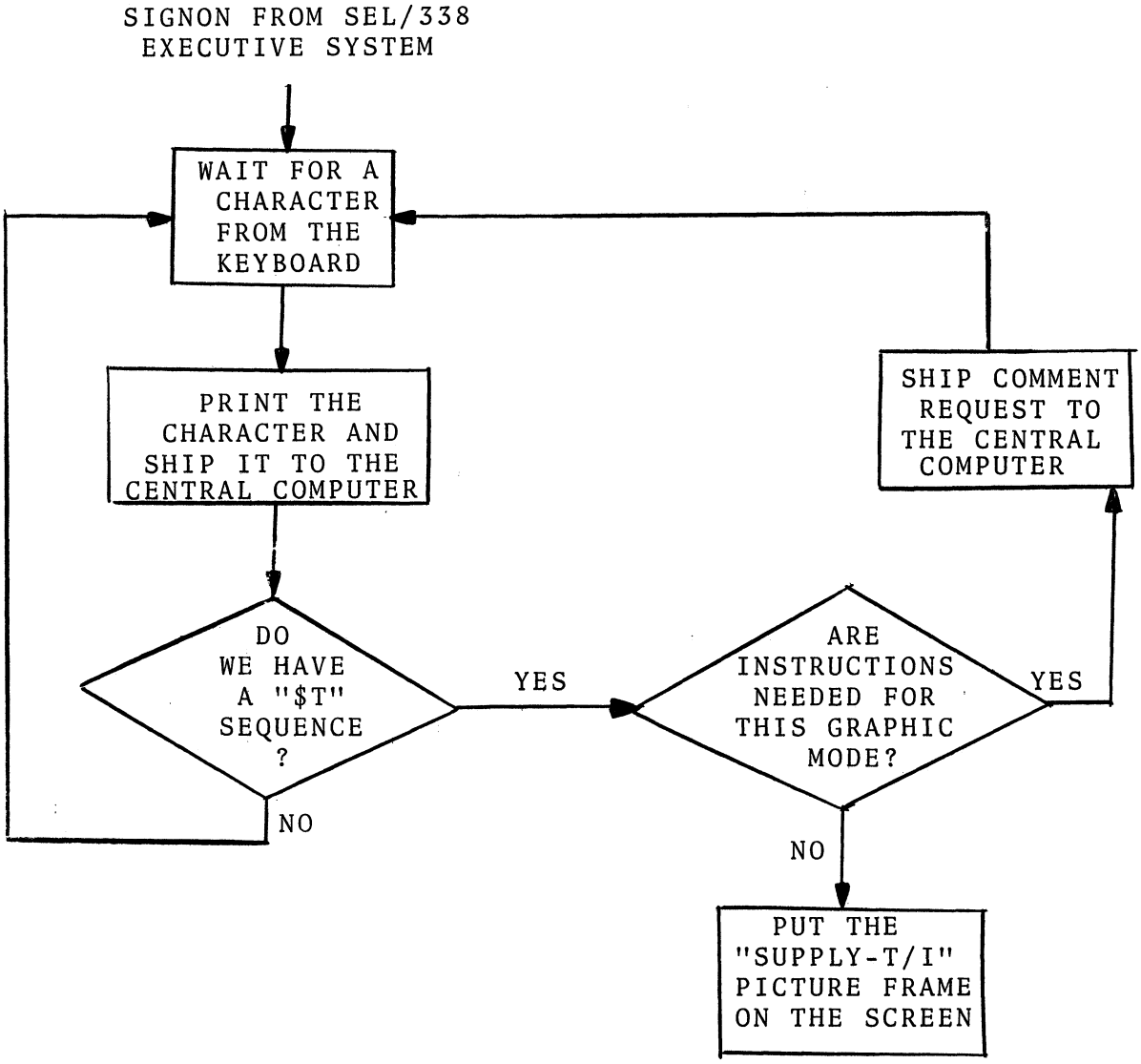
```
XK      DC    K       ACCEPT OCTAL CHARACTER FROM KB (F)
XBINI   DC    BINI    ACCEPT OCTAL FROM KEYBOARD (F)
XBINO   DC    BINO    TYPE OCTAL
XCRLF   DC    CRLF    TELEPRINTER CARRIAGE POSITIONING
XLIB    DC    LIB     FEED DISPLAY TEXT BUFFER (F)
XLSC    DC    LSC     CLEAR DISPLAY STRUCTURES
XLSL    DC    LSL     CREATE NEW LEVEL (F)
XLSD    DC    LSD     DESTROY LEVEL (F)
XLSI    DC    LSI     INSERT SUB-STRUCTURE INTO LEVEL (F)
XLSIR   DC    LSIR    RELATIONAL INSERT LEVEL INTO LEVEL (F)
XLSR    DC    LSR     REMOVE SUB-STRUCTURE FROM LEVEL (F)
XLSRR   DC    LSRR    RELATIONAL REMOVE LEVEL FROM LEVEL (F)
XLMY    DC    LMY     TRANSLATE LEVEL IN 'Y' DIRECTION
XLMX    DC    LMX     TRANSLATE LEVEL IN 'X' DIRECTION
XLMP    DC    LMP     SET LEVEL PARAMETERS
XLMBN   DC    LMBN    BLINK LEVEL
XLMBF   DC    LMBF    STOP LEVEL BLINK
XLMC    DC    LMC     COUNT LEVEL PARAMETERS
XLMU    DC    LMU     STOP UNCONDITIONALLY AT END OF LEVEL
XLMN    DC    LMN     DO NOT STOP AT END OF LEVEL
XLMS    DC    LMS     STOP AT END OF LEVEL IF ON SCREEN
XLML    DC    LML     STOP AT END OF LEVEL ON LPSI
XW1     DC    W1      SCHEDULE PREVIOUS LOCATION AS A TASK
XW2     DC    W2      SCHEDULE SECOND PREVIOUS LOC. AS A TASK
XAS     DC    AS      CREATE A 4N (N=NO. OF 4 WORD BLOCKS)
XCDTC   DC    CDTC    CONVERT DISPLAY TO TWO'S COMPLEMENT
XCTDC   DC    CTDC    CONVERT TWO'S COMPLEMENT TO DISPLAY
XBFIC   DC    BFIC    CLEAR 103 INPUT BUFFER
XBFIM   DC    BFIM    GET CHARACTER FROM 103 INPUT BUFFER
XBFOC   DC    BFOC    CLEAR 103 OUTPUT BUFFER
XBFOM   DC    BFOM    SEND CHARACTER TO 103 OUTPUT BUFFER
XCC     DC    CC      FUNCTION EXTENSION
XIR     DC    IR      INTERRUPT RETURN
```

Table A3.1 SEL/338 Transfer Vector

A3.2 TDS/338

The peripheral portion of TDS acts just like an on-line teletype except in the "$T" mode, that is, when problem topologies are being inputted graphically.

Hence, the control section flow chart is shown in Figure A3.1.



NOTE:
INFORMATION FROM THE CENTRAL COMPUTER
GOES DIRECTLY TO THE TELETYPE OR SCREEN
VIA A "RECEIVE" ROUTINE.

Figure A3.1. TDS/338 Control Section Flow Chart.

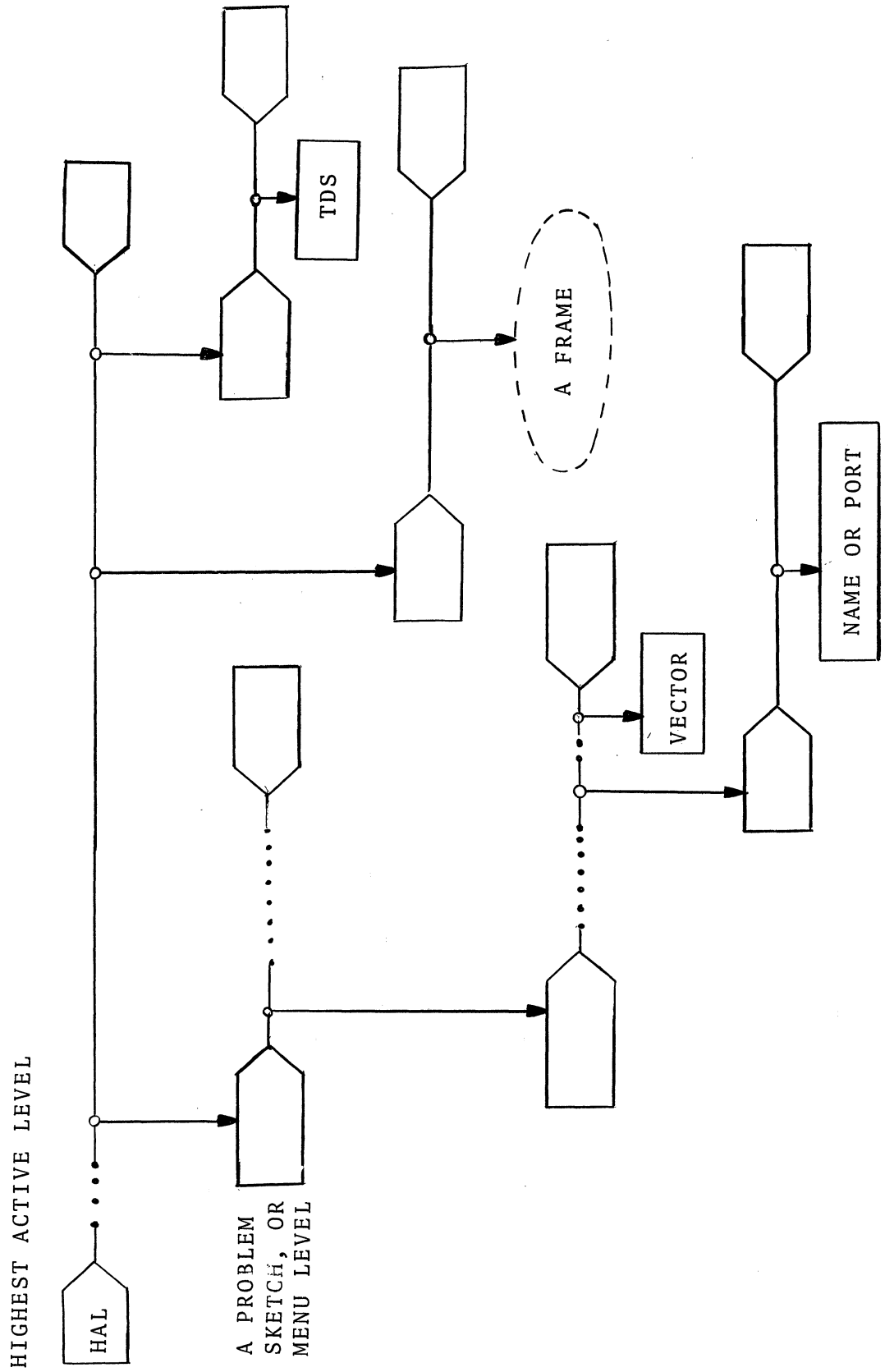


Figure A3.2. General Scheme of User Graphic Storage (Pheripheral Data Structure).

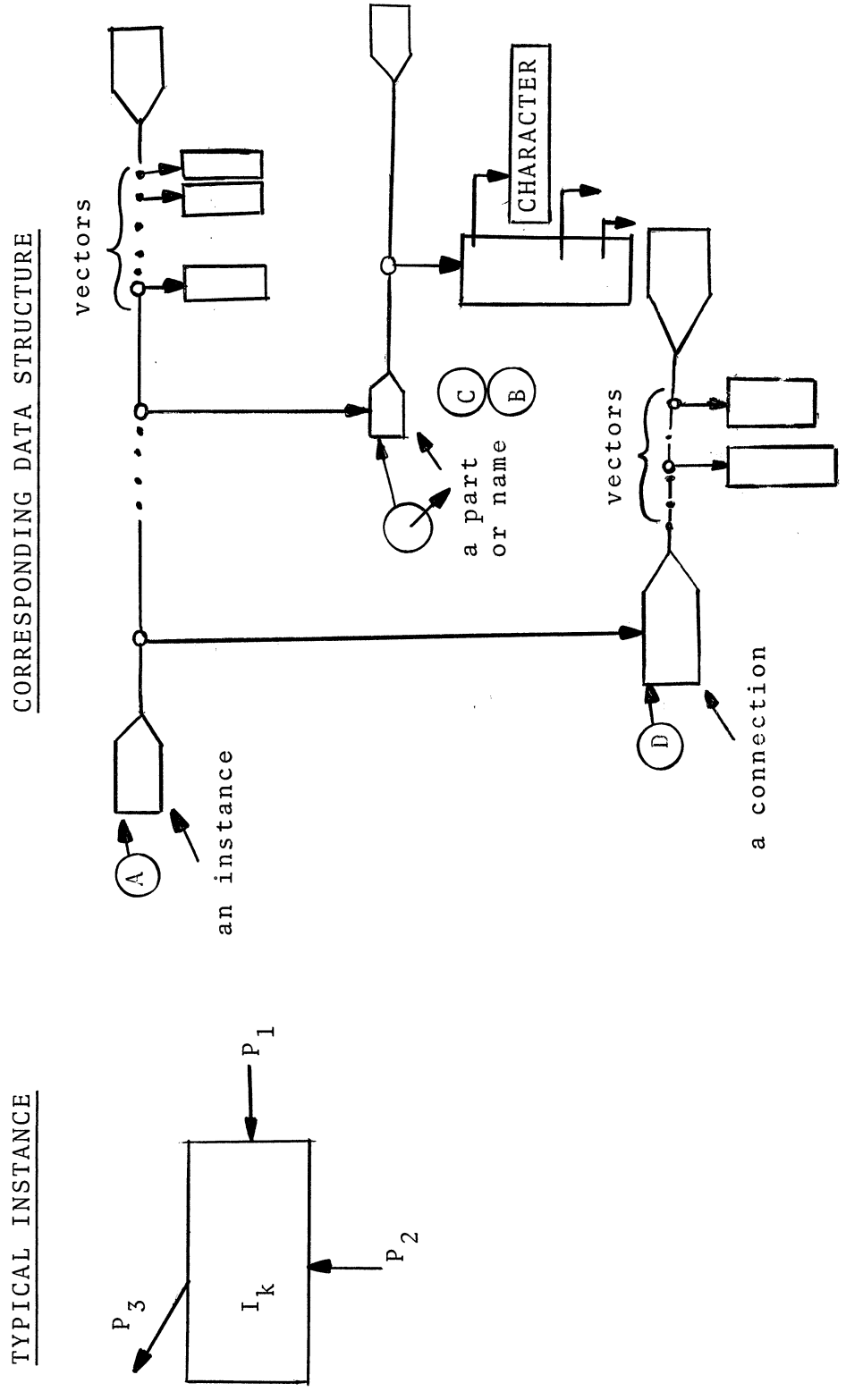


Figure A3.3. A Typical Graphic Instance and some of the Corresponding Data Structure.

N-TUPLE KIND	FORM	EXAMPLE
NAMES	N A - B : 6 CHARS.	N4060-4120:COMPR5
PORTS	P A I/O/P/T C : 6 CHARS.	P4060T4160:H1
DELETED PORTS	D A - C	D4060-4160
CONNECTIONS	M C - C' : C	M4250-5340:5670
DISCONNECTIONS	B D	B5670
CALLS	C A	C4060
REMOVALS	R A	R4060

Table A3.2 N-Tuples Transmitted by the Graphic Routines to the Central Computer

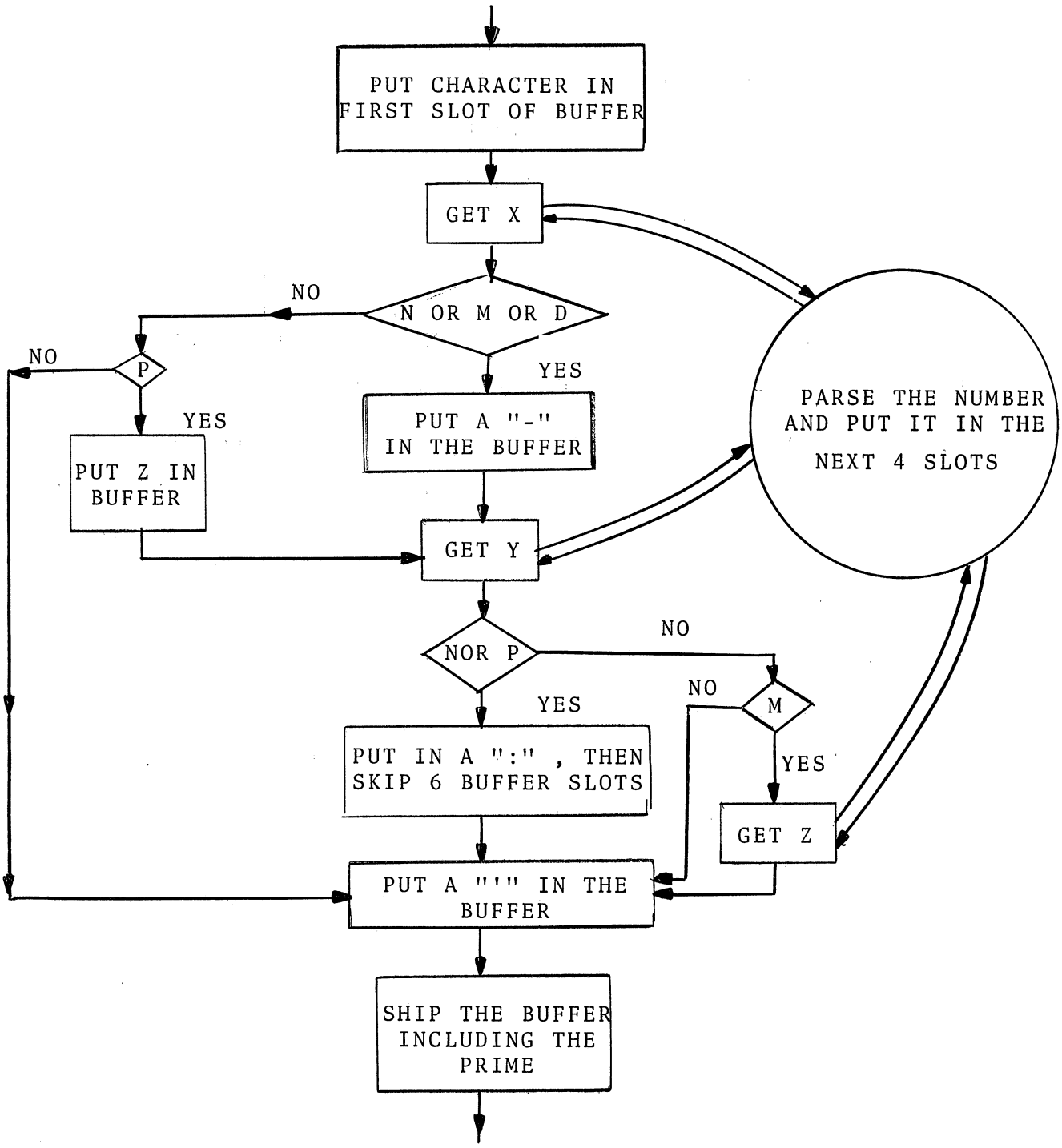


Figure A3.4. Flow Chart of Routine to Transmit N-Tuples.

The general scheme of user storage in the peripheral computer is shown in Figure A3.2. This depicts the graphic data structure in conceptual fashion. For details of the "home plates" and the "nodes" see the forthcoming Systems Engineering Laboratory Report.[5]

Figure A3.3 shows a typical graphic instance and some of the corresponding data structure. The purpose of the figure is to show that unique addresses are the "keys" to topology interpretation. Table A3.2 lists the ordered n-tuples that are shipped, and which are necessary and sufficient to connote a current topology. The shipping routine is flow charted in Figure A3.4.

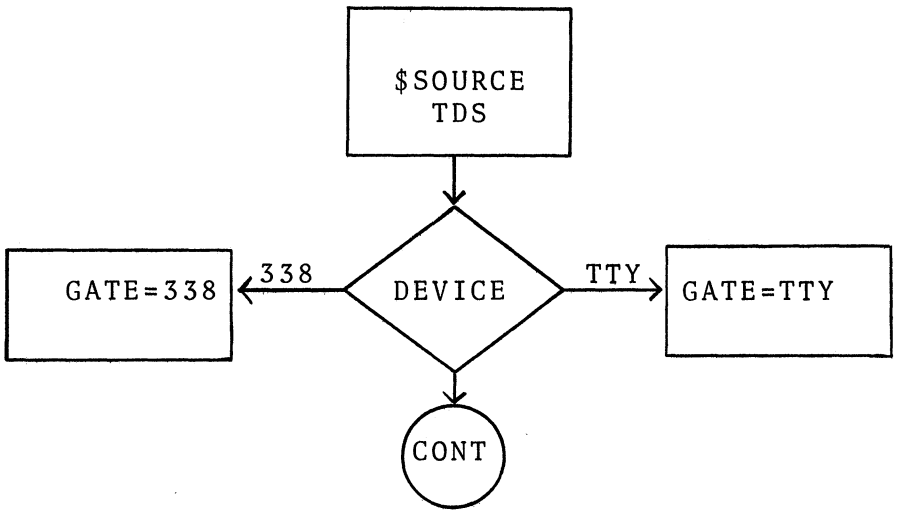
A3.3 TDS/360

Once the designer is signed on to the central portion of TDS, the system is always at his disposal. This means that the following TDS flow charts are of "closed" form.

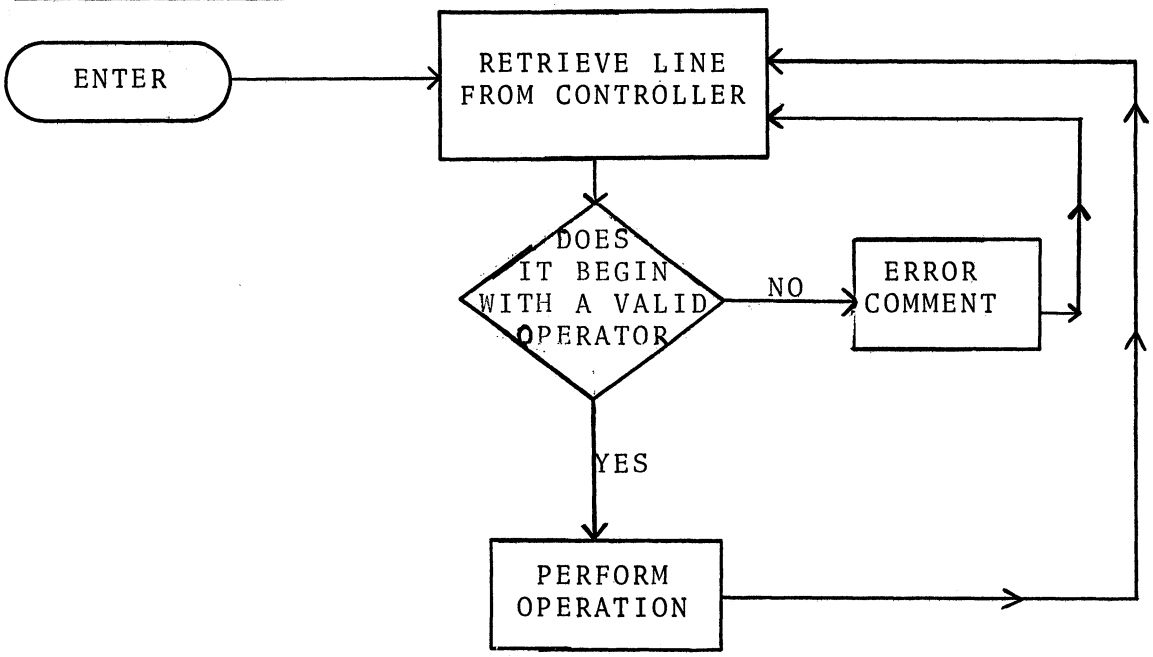
---

[5] James H. Jackson's report on the SEL Executive, which should be released in the summer of 1968.

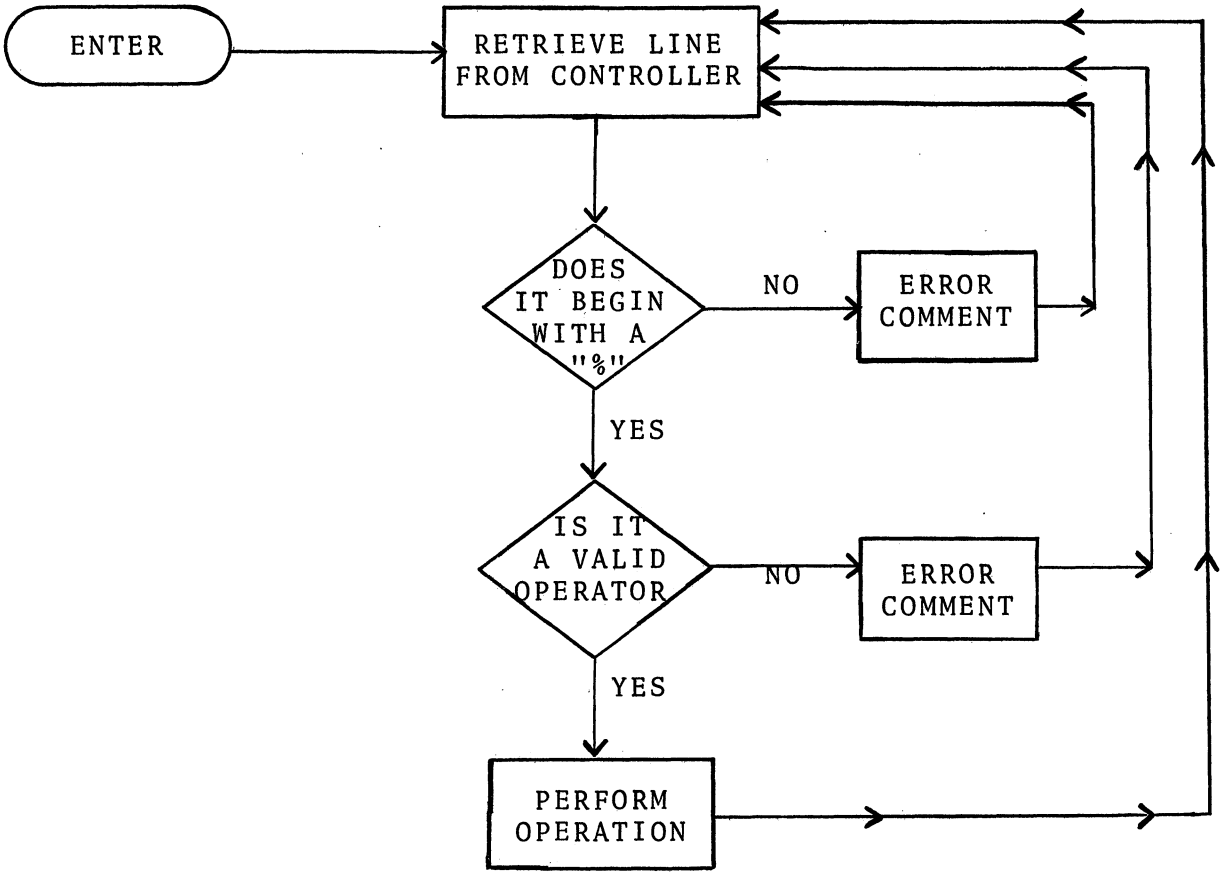
$SOURCE



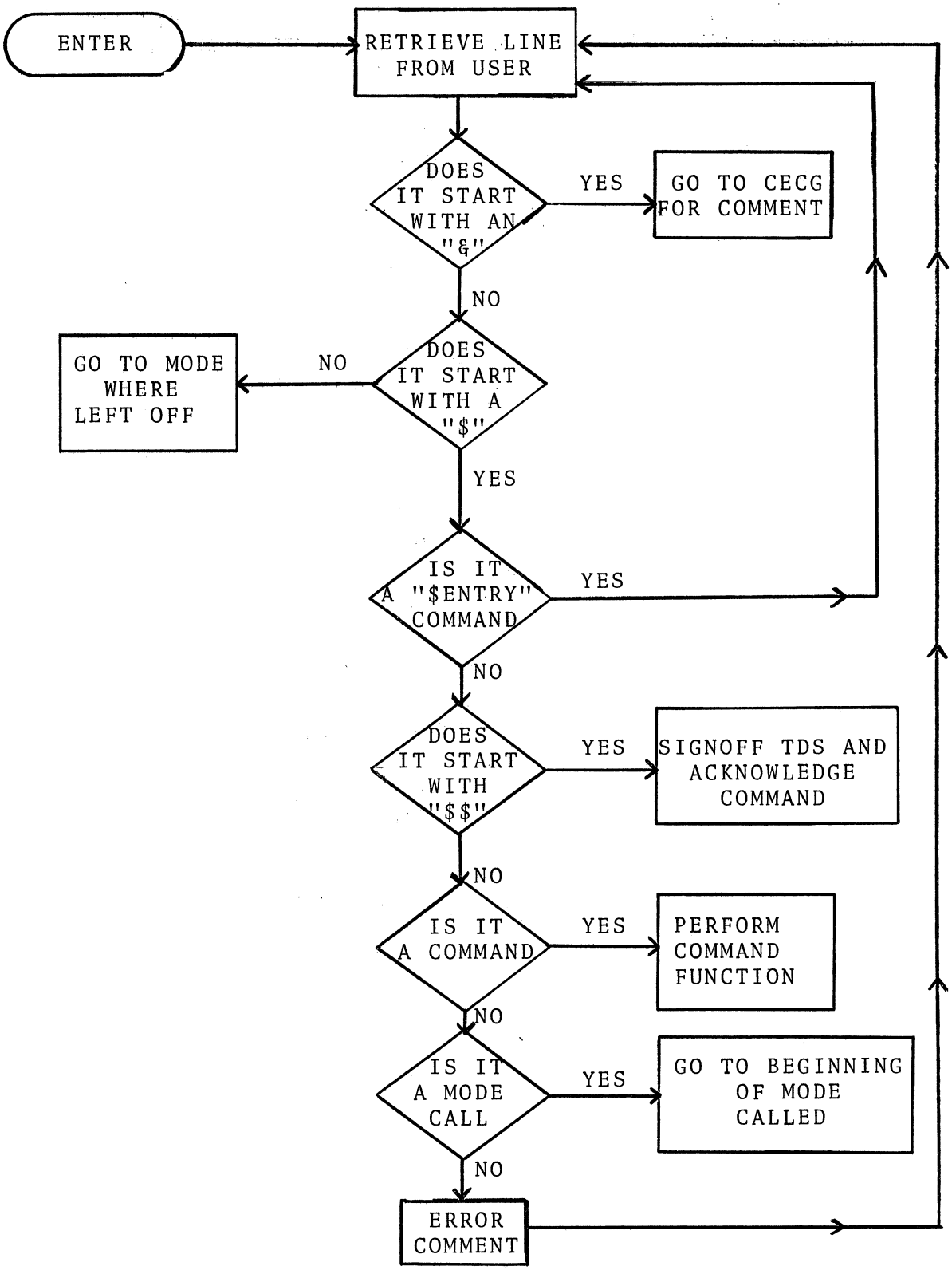
T/I (GATE=338)



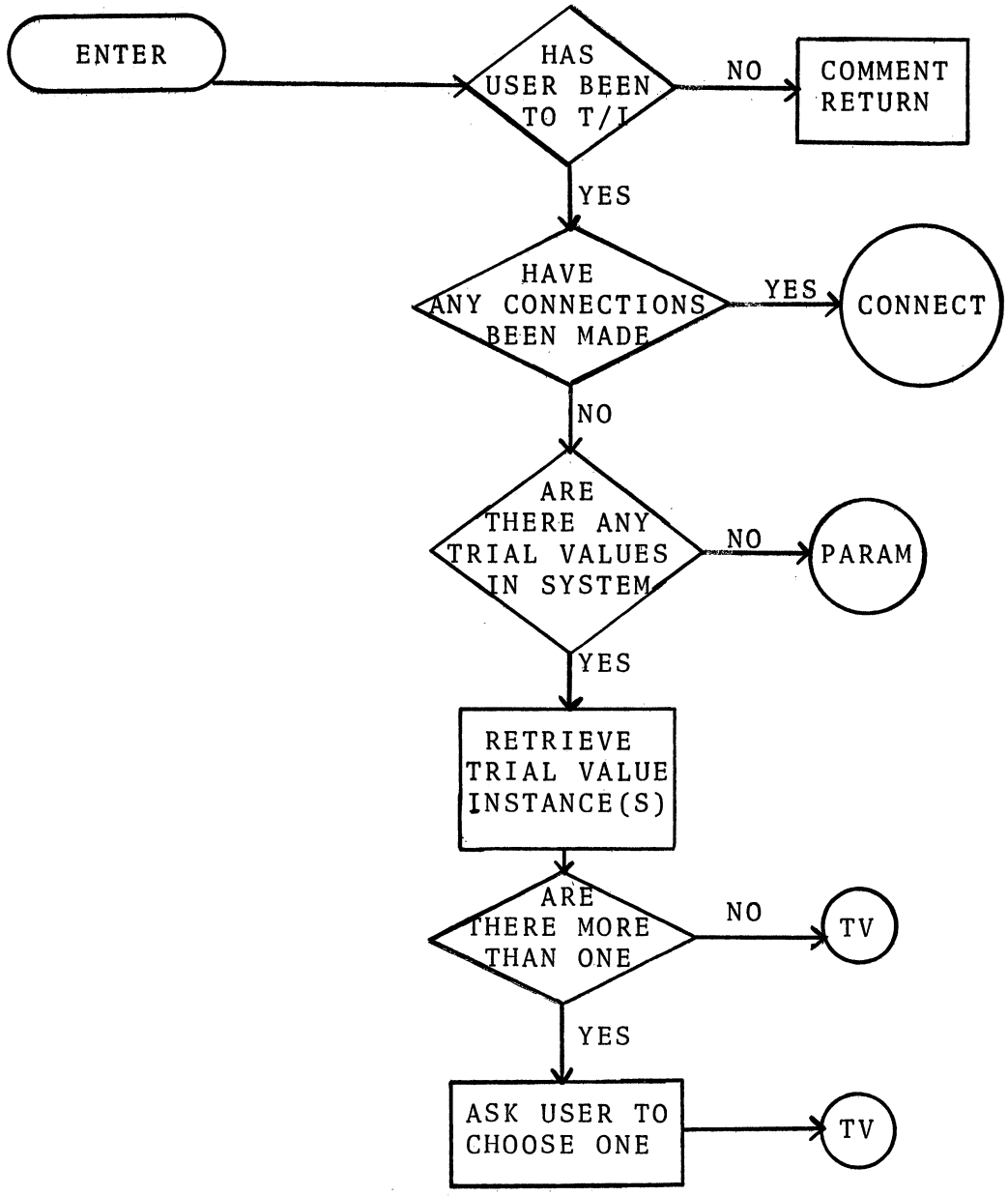
T/I (GATE = TTY)


ENTER
RETRIEVE LINE FROM CONTROLLER
DOES IT BEGIN WITH A "%"
NO
ERROR COMMENT
YES
IS IT A VALID OPERATOR
NO
ERROR COMMENT
YES
PERFORM OPERATION

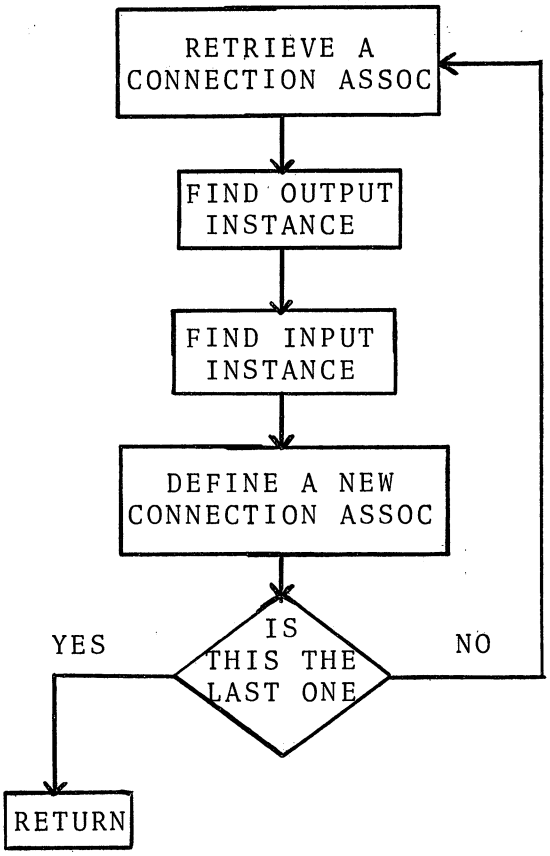
CONTROLLER

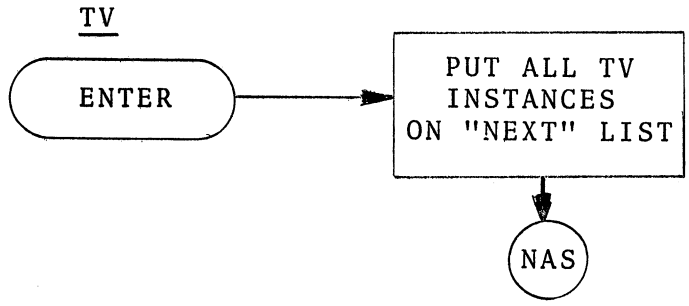

ENTER
RETRIEVE LINE FROM USER
DOES IT START WITH AN "&"
YES
GO TO CECG FOR COMMENT
NO
DOES IT START WITH A "$"
NO
GO TO MODE WHERE LEFT OFF
YES
IS IT A "$ENTRY" COMMAND
YES
NO
DOES IT START WITH "$$"
YES
SIGNOFF TDS AND ACKNOWLEDGE COMMAND
NO
IS IT A COMMAND
YES
PERFORM COMMAND FUNCTION
NO
IS IT A MODE CALL
YES
GO TO BEGINNING OF MODE CALLED
NO
ERROR COMMENT

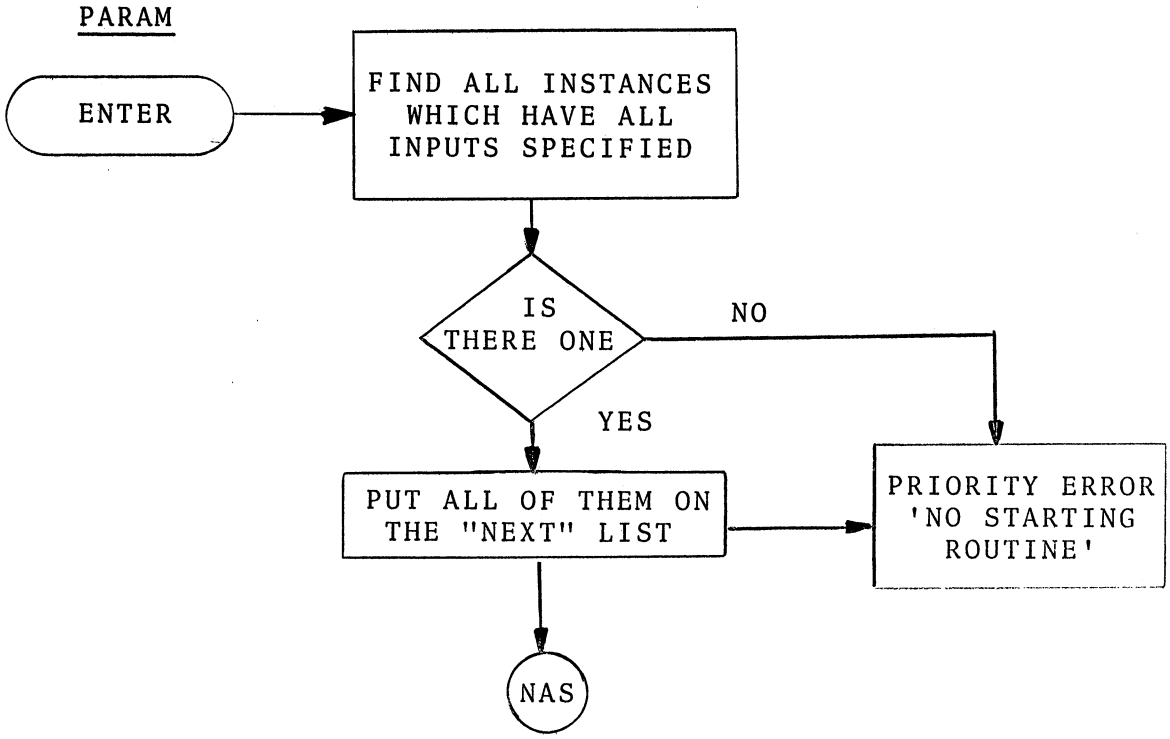
BUILD

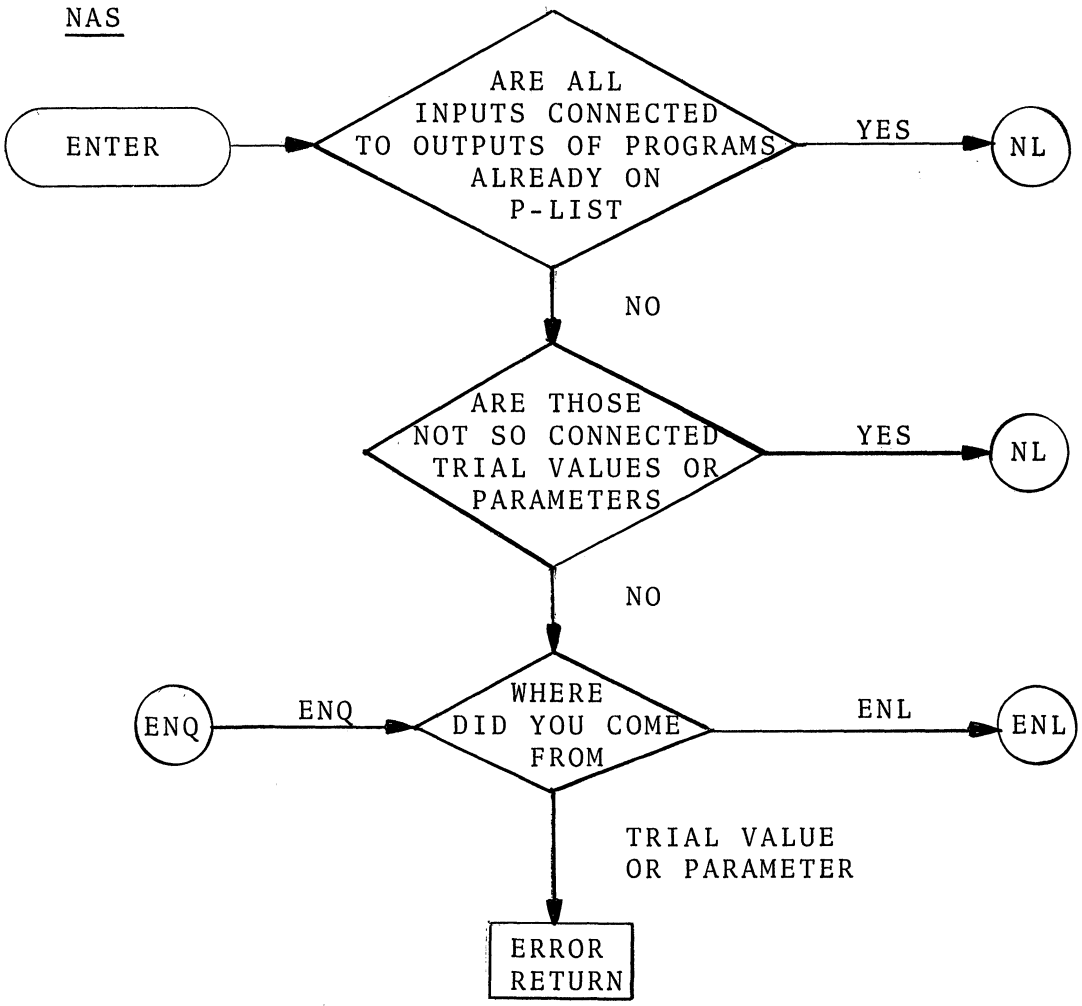

ENTER
HAS USER BEEN TO T/I
NO
COMMENT RETURN
YES
HAVE ANY CONNECTIONS BEEN MADE
YES
CONNECT
NO
ARE THERE ANY TRIAL VALUES IN SYSTEM
NO
PARAM
YES
RETRIEVE TRIAL VALUE INSTANCE(S)
ARE THERE MORE THAN ONE
NO
TV
YES
ASK USER TO CHOOSE ONE
TV

CONNECT

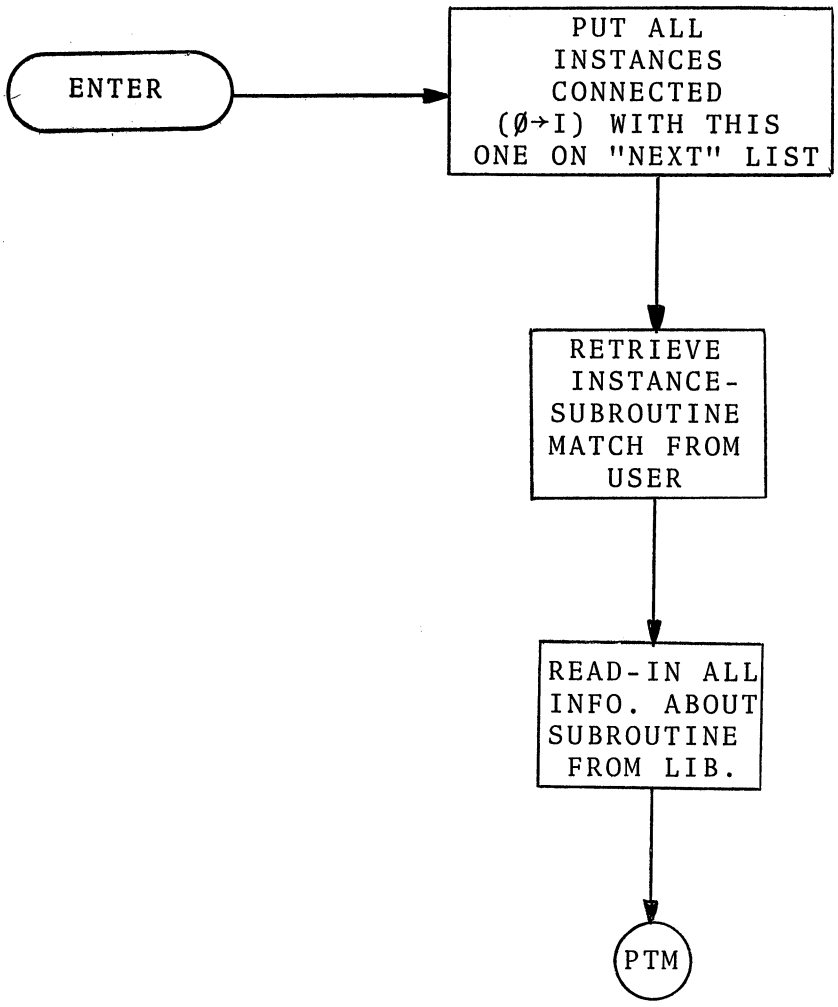

RETRIEVE A CONNECTION ASSOC
FIND OUTPUT INSTANCE
FIND INPUT INSTANCE
DEFINE A NEW CONNECTION ASSOC
IS THIS THE LAST ONE
YES
NO
RETURN


TV
ENTER
PUT ALL TV INSTANCES ON "NEXT" LIST
NAS

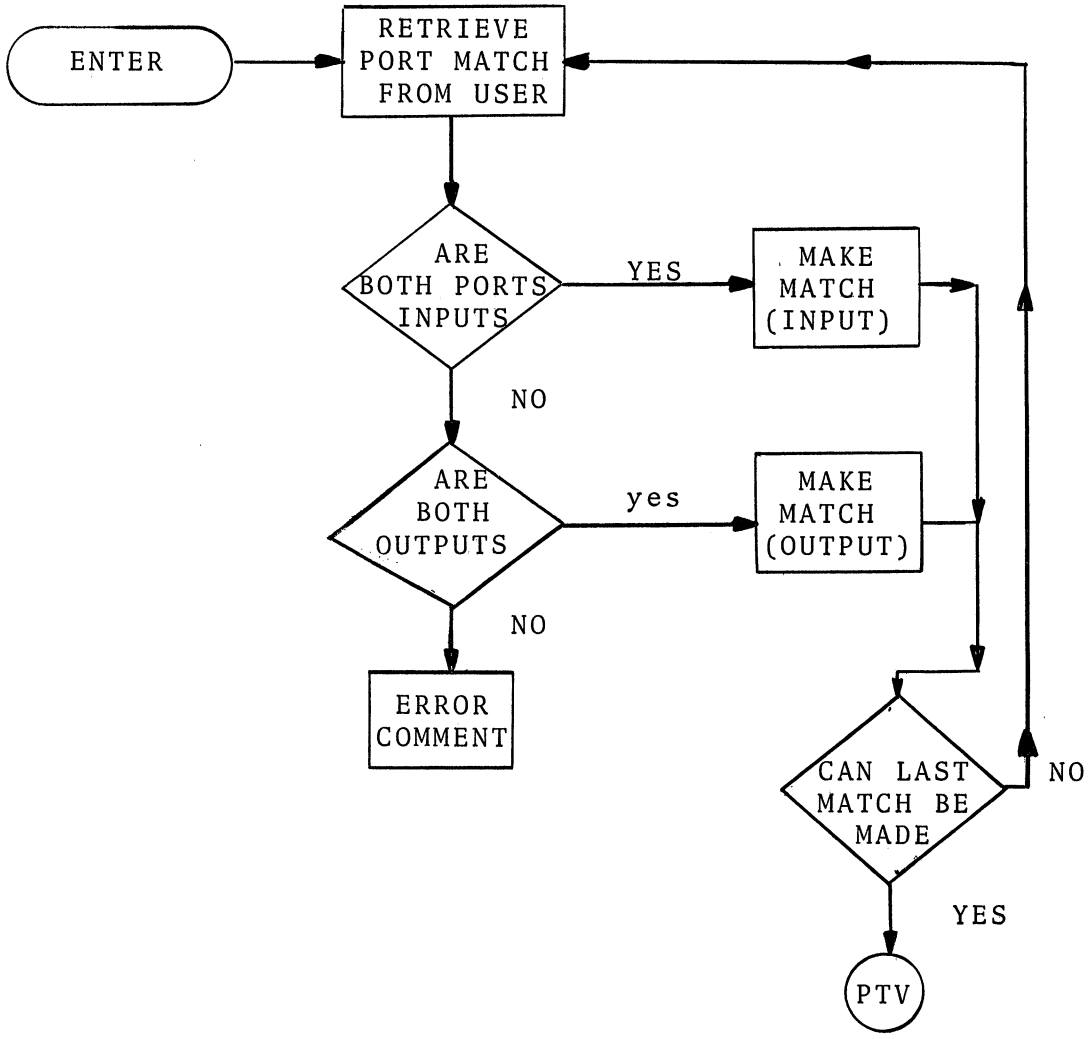

PARAM
ENTER
FIND ALL INSTANCES WHICH HAVE ALL INPUTS SPECIFIED
IS THERE ONE
NO
YES
PUT ALL OF THEM ON THE "NEXT" LIST
PRIORITY ERROR 'NO STARTING ROUTINE'
NAS


NAS
ENTER
ARE ALL INPUTS CONNECTED TO OUTPUTS OF PROGRAMS ALREADY ON P-LIST
YES
NL
NO
ARE THOSE NOT SO CONNECTED TRIAL VALUES OR PARAMETERS
YES
NL
NO
ENQ
ENQ
WHERE DID YOU COME FROM
ENL
ENL
TRIAL VALUE OR PARAMETER
ERROR RETURN

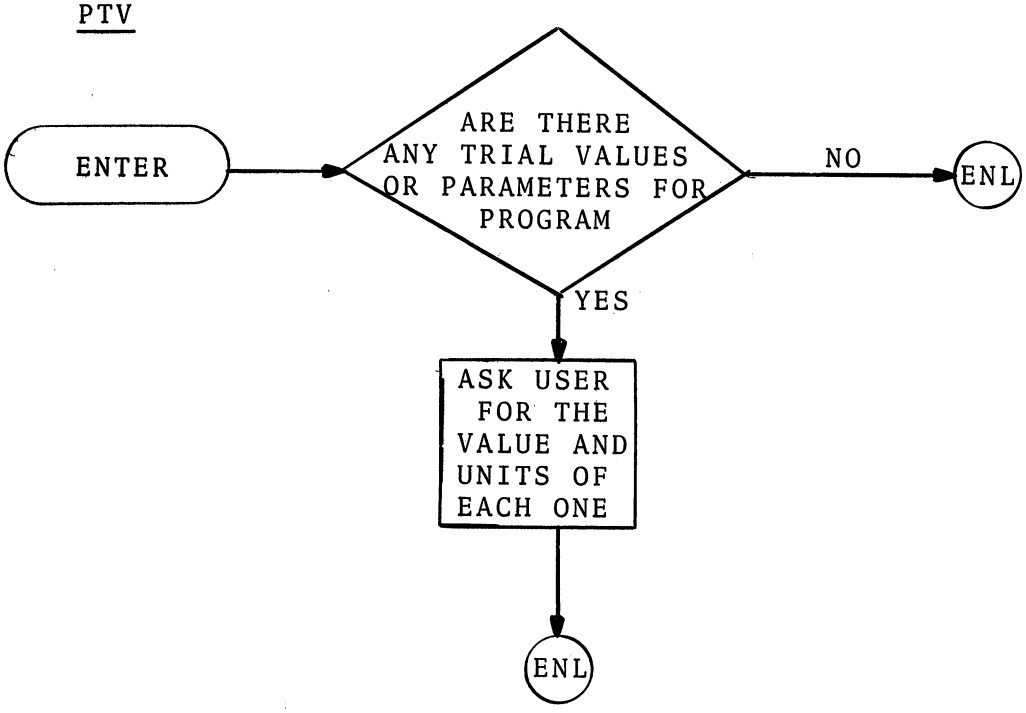
NL


ENTER
PUT ALL INSTANCES CONNECTED (Ø→I) WITH THIS ONE ON "NEXT" LIST
RETRIEVE INSTANCE-SUBROUTINE MATCH FROM USER
READ-IN ALL INFO. ABOUT SUBROUTINE FROM LIB.
PTM

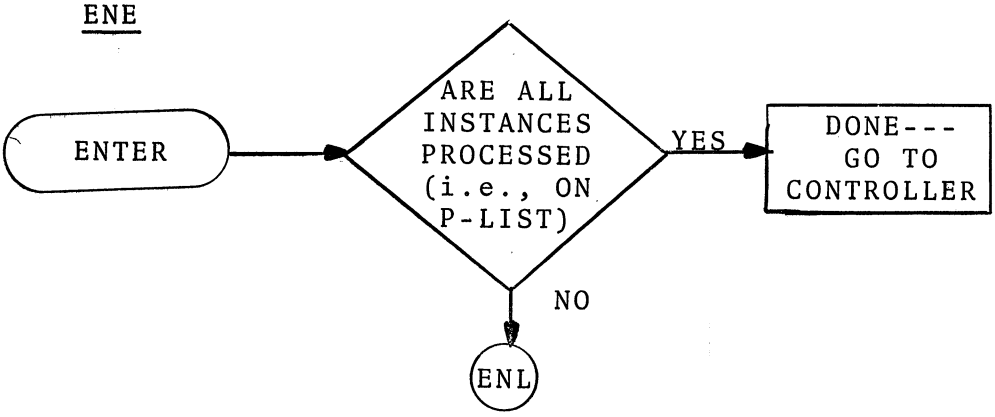
PTM


ENTER
RETRIEVE PORT MATCH FROM USER
ARE BOTH PORTS INPUTS
YES
MAKE MATCH (INPUT)
NO
ARE BOTH OUTPUTS
yes
MAKE MATCH (OUTPUT)
NO
ERROR COMMENT
CAN LAST MATCH BE MADE
NO
YES
PTV

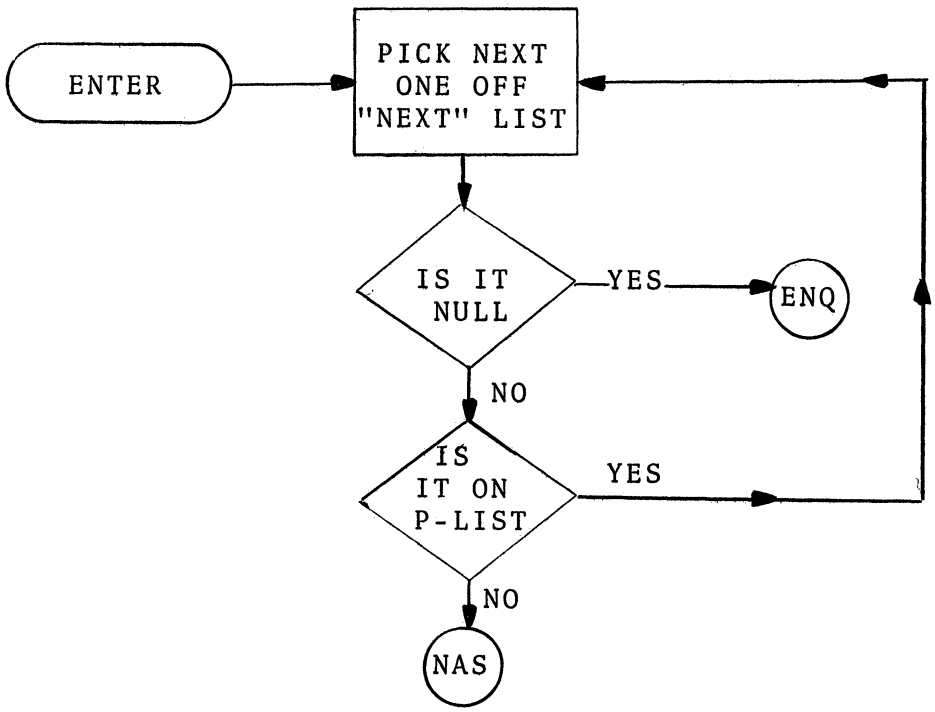
PTV

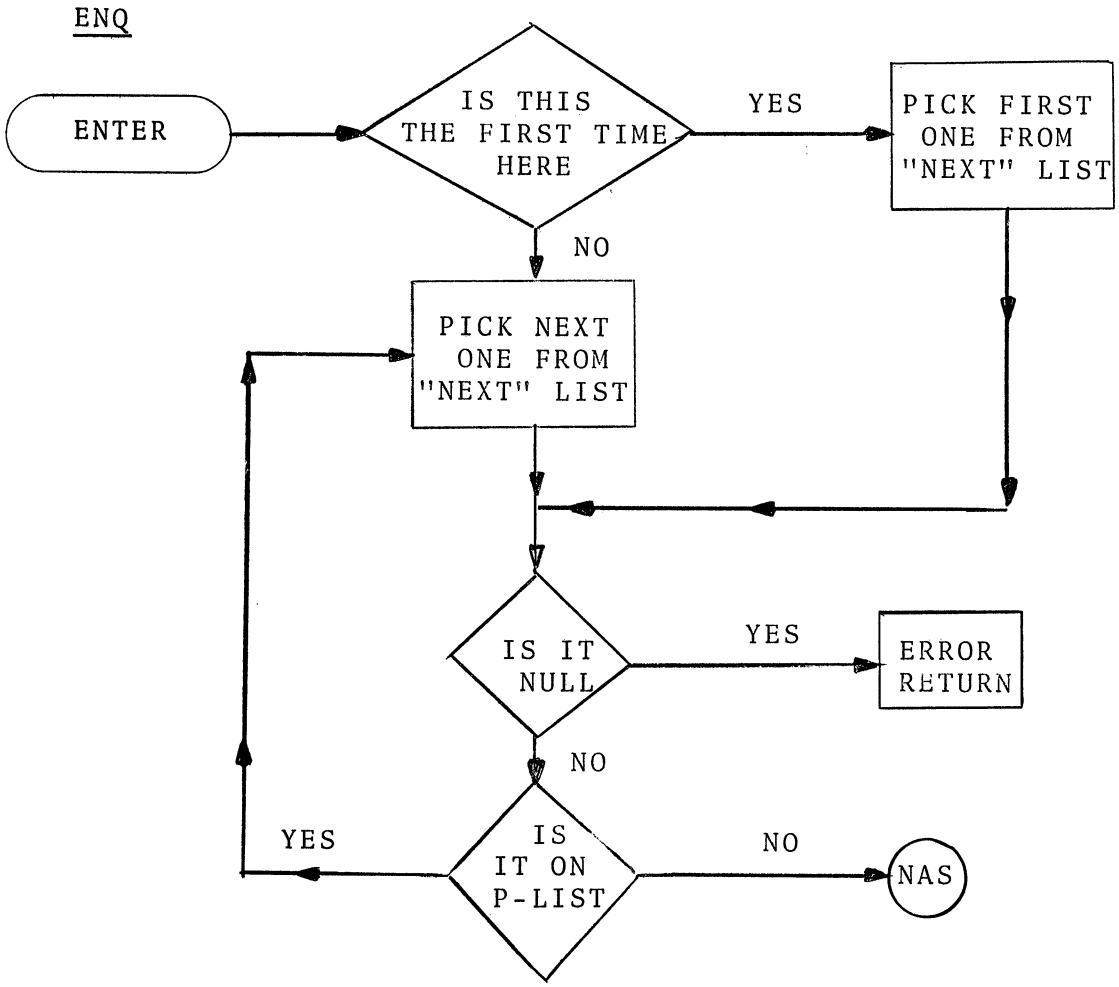

ENTER
ARE THERE ANY TRIAL VALUES OR PARAMETERS FOR PROGRAM
NO
ENL
YES
ASK USER FOR THE VALUE AND UNITS OF EACH ONE
ENL

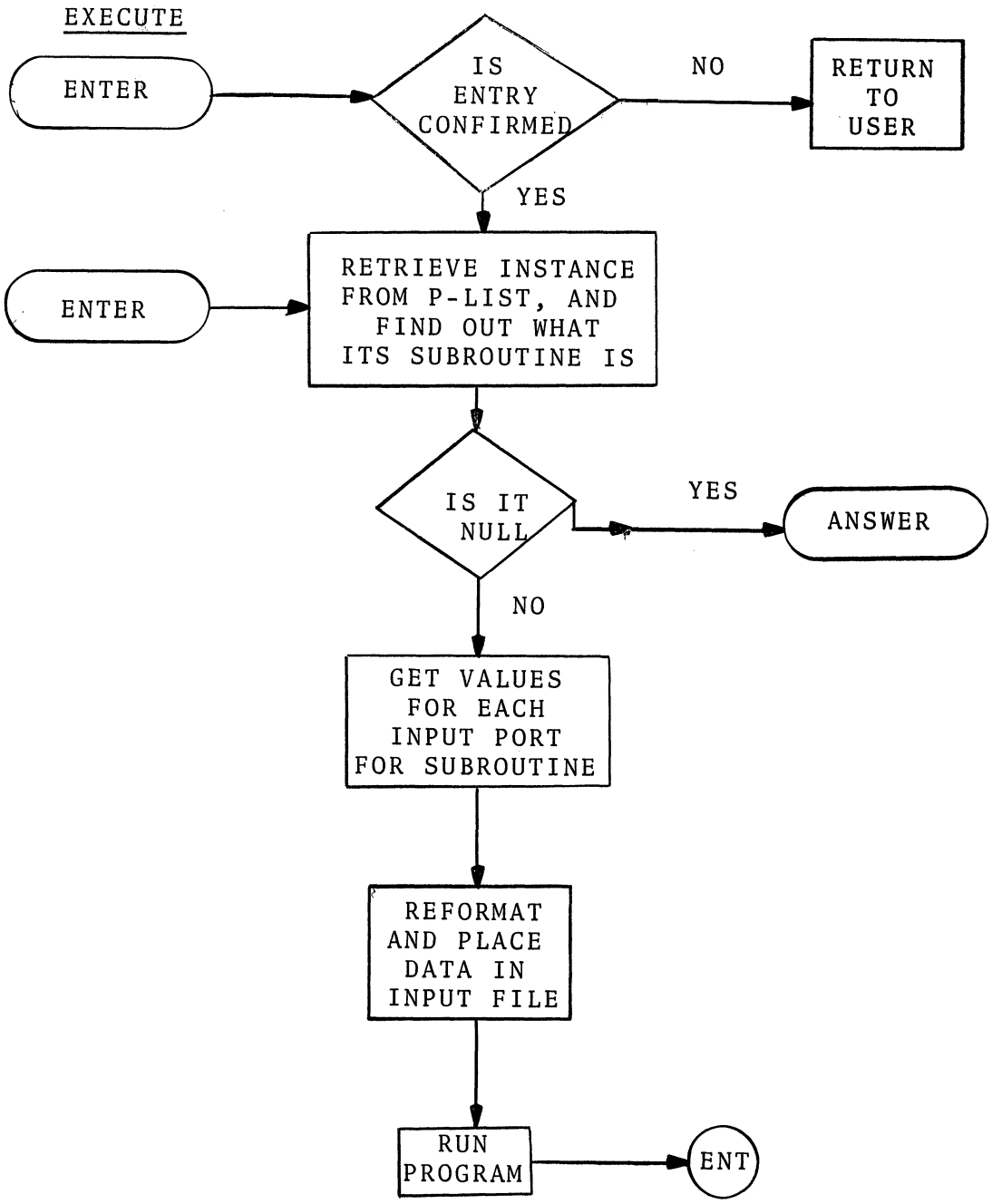
ENE

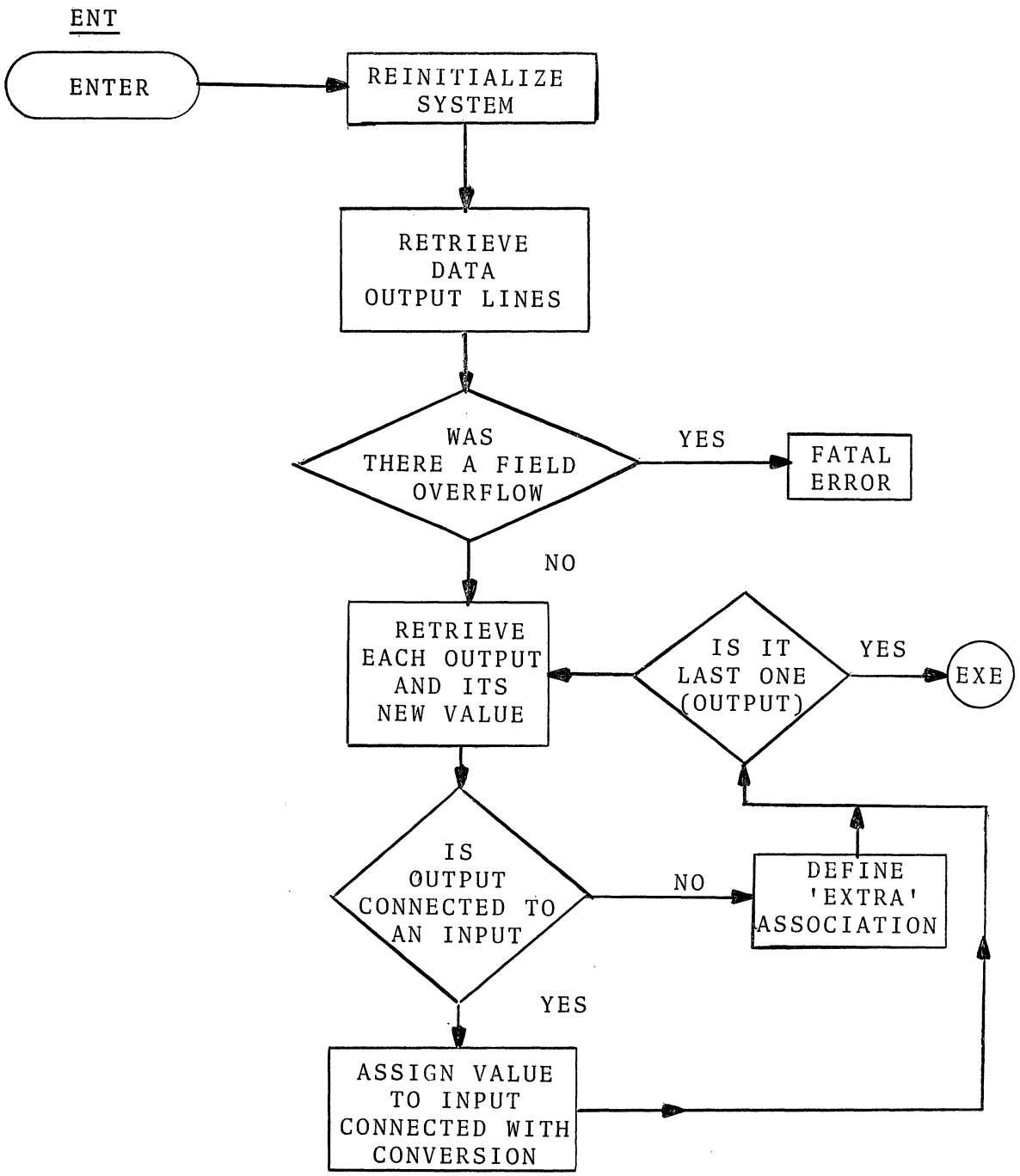

ENTER
ARE ALL INSTANCES PROCESSED (i.e., ON P-LIST)
YES
DONE--- GO TO CONTROLLER
NO
ENL

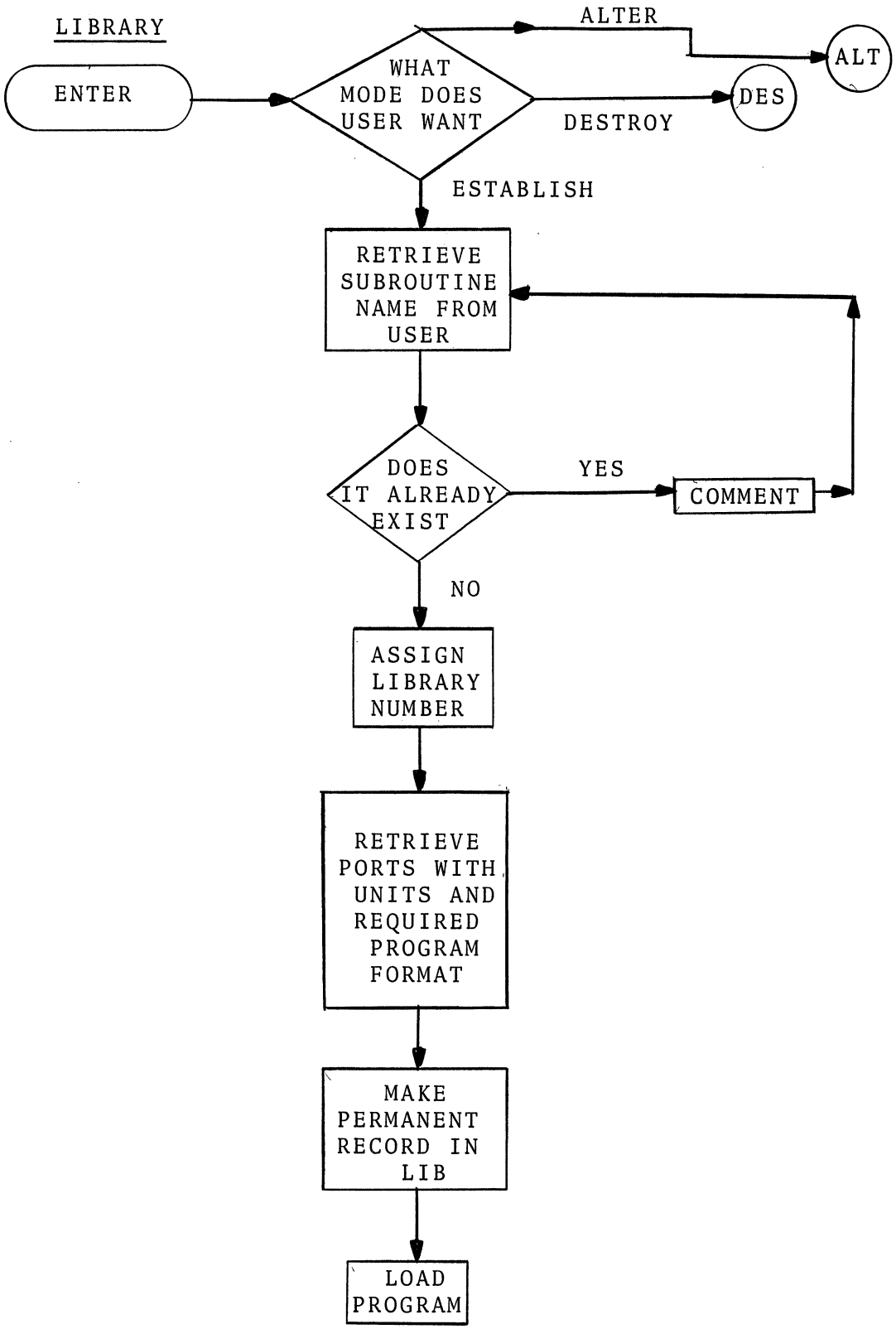
ENL


ENTER
PICK NEXT ONE OFF "NEXT" LIST
IS IT NULL
YES
ENQ
NO
IS IT ON P-LIST
YES
NO
NAS

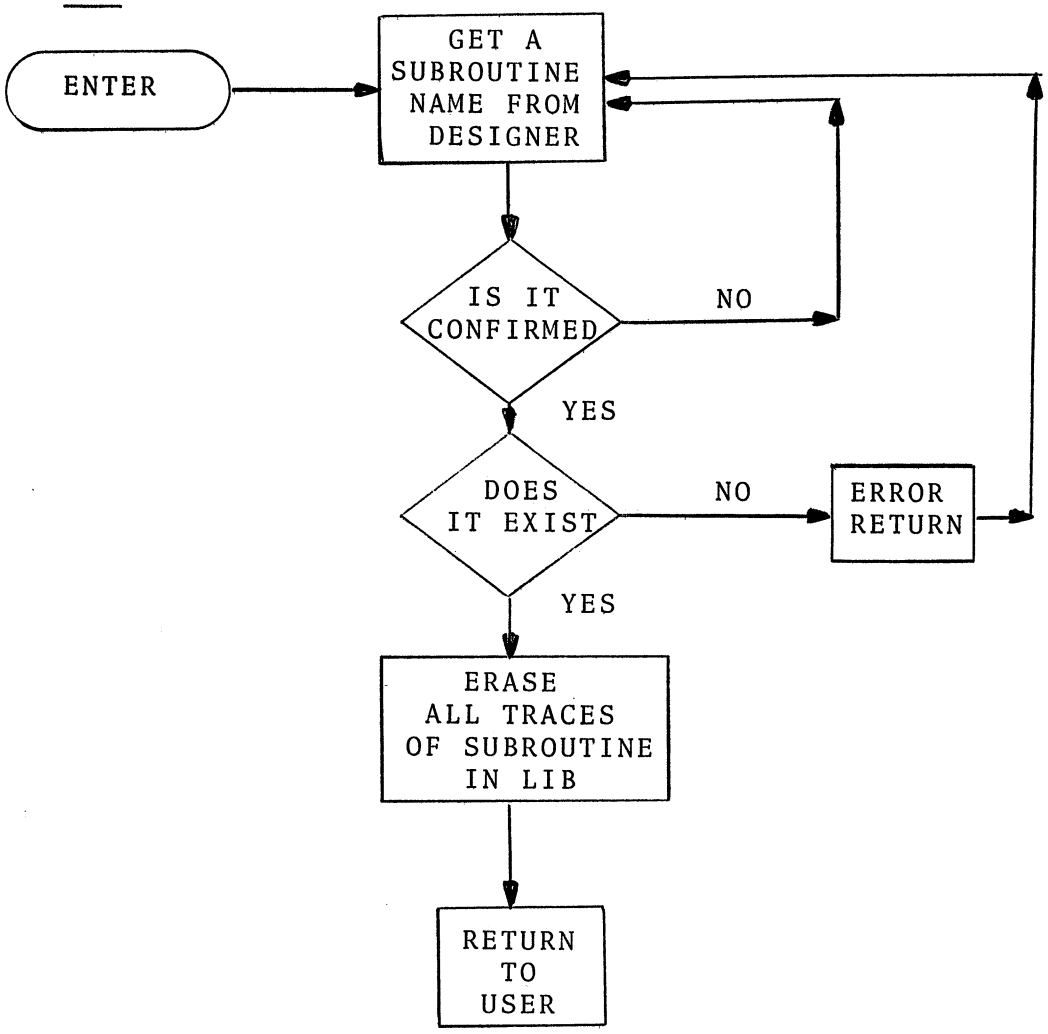

ENQ
ENTER
IS THIS THE FIRST TIME HERE
YES
PICK FIRST ONE FROM "NEXT" LIST
NO
PICK NEXT ONE FROM "NEXT" LIST
IS IT NULL
YES
ERROR RETURN
NO
IS IT ON P-LIST
YES
NO
NAS

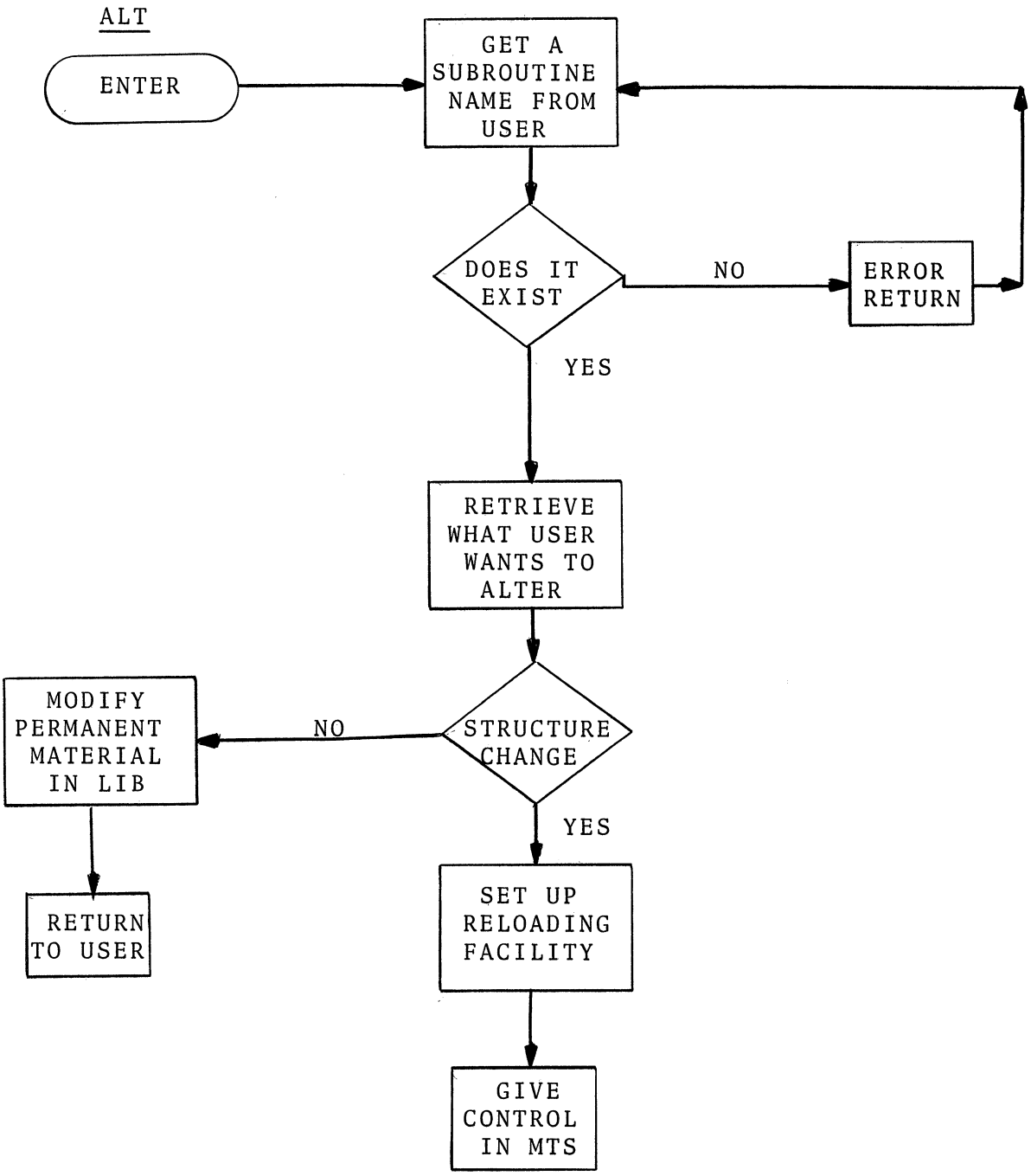

EXECUTE
ENTER
IS ENTRY CONFIRMED
NO
RETURN TO USER
YES
ENTER
RETRIEVE INSTANCE FROM P-LIST, AND FIND OUT WHAT ITS SUBROUTINE IS
IS IT NULL
YES
ANSWER
NO
GET VALUES FOR EACH INPUT PORT FOR SUBROUTINE
REFORMAT AND PLACE DATA IN INPUT FILE
RUN PROGRAM
ENT


ENT
ENTER
REINITIALIZE SYSTEM
RETRIEVE DATA OUTPUT LINES
WAS THERE A FIELD OVERFLOW
YES
FATAL ERROR
NO
RETRIEVE EACH OUTPUT AND ITS NEW VALUE
IS IT LAST ONE (OUTPUT)
YES
EXE
IS OUTPUT CONNECTED TO AN INPUT
NO
DEFINE 'EXTRA' ASSOCIATION
YES
ASSIGN VALUE TO INPUT CONNECTED WITH CONVERSION


LIBRARY
ENTER
WHAT MODE DOES USER WANT
ALTER
ALT
DESTROY
DES
ESTABLISH
RETRIEVE SUBROUTINE NAME FROM USER
DOES IT ALREADY EXIST
YES
COMMENT
NO
ASSIGN LIBRARY NUMBER
RETRIEVE PORTS WITH UNITS AND REQUIRED PROGRAM FORMAT
MAKE PERMANENT RECORD IN LIB
LOAD PROGRAM

DES


ENTER
GET A SUBROUTINE NAME FROM DESIGNER
IS IT CONFIRMED
NO
YES
DOES IT EXIST
NO
ERROR RETURN
YES
ERASE ALL TRACES OF SUBROUTINE IN LIB
RETURN TO USER


ALT
ENTER
GET A SUBROUTINE NAME FROM USER
DOES IT EXIST
NO
ERROR RETURN
YES
RETRIEVE WHAT USER WANTS TO ALTER
STRUCTURE CHANGE
NO
MODIFY PERMANENT MATERIAL IN LIB
RETURN TO USER
YES
SET UP RELOADING FACILITY
GIVE CONTROL IN MTS

APPENDIX 4

DIVISION OF LABOR BETWEEN THE PERIPHERAL AND MAIN COMPUTERS

As developed in the main text of this dissertation, the primary purpose for a computer-based environment for the engineering designer is so that he can analyze models of real physical systems more readily, thus enabling more quantitative considerations of component interactions in a given problem. Furthermore, it was pointed out that one of the three categories of service provided by his computer-based environment is "analysis," another was information retrieval, and a third was teaching. One general conclusion is that in all cases a large amount of storage is apparently needed for the designer. Hence, this requires a large computer in a time-shared mode, or a fairly large slave computer. Because of the hardware available to the author, a System/360, Model 67 computer was used as the central computer, and a DEC 338 as the peripheral computer. The idea of having a peripheral computer at all is so that the topology of a design problem can be manipulated graphically. The potential value of this was one of the premises for embarking on this work, and the value of it has been demonstrated by the working system. Consequently, the large central computer is conceived for all things where much computing and much storage are needed, and the peripheral computer is conceived as the element that gives the fast interaction and the graphic capability. The current division of graphics would probably be changed in a new version of TDS. Because of the limited size of the 338 memory, a new version would

probably keep the expanded data structure in the central computer because of the large storage, and would keep a concatenated version in the 338. This would enable many more symbols to be on the screen at one time. One assumption here is that the data link between computers would be fast enough so as not to ruin the human factors involved in the drawing. As it is now, all the speed that is desired is available in the graphic mode, and it fulfills its design requirements. However, the change would be made if the speed would still be fast enough after getting the trade-off for storage capacity.

There are no tasks currently performed in the central computer that should be moved to the peripheral computer. However, within the peripheral computer, because of the banked structure of the hardware, there would be a new allocation of tasks between banks. This can hardly be taken as a general principle, in that many designers of systems using satellite and central computers have a satellite with a large contiguous core.

APPENDIX 5

A QUANTITATIVE DISCUSSION OF EXCESS INFORMATION

Consider that involved in both the representations of Figures 3.1 and 3.2 is the following subset of geometrical notions: straight lines, equal length lines, right angles, horizontal and vertical lines. Further, consider that to represent the square in computer instructions, the designer must also use the following subset of computer notions: iteration, labels, modes, intensity and storage. If "use" is taken to mean "being able to spontaneously employ the notion creatively," then the probability of a system designer "using" a concept can itself be used to calculate a value for the information generated by the designer. Note that this latter "information" is <u>not</u> that which conveys the square to the computer, but that which would convey "to an observer" the designer's own personal store of information.

Now, as a designer creates a square via either of the methods depicted in Figures 3.1 or 3.2, the following statement must be made so that the information involved can be handled quantitatively. Being able to "use" the above geometrical notions, is, for a designer, <u>independent</u> of his being able to "use" the computer notions. Formally this is stated as:

*** Proposition A5.1

The probability that a designer can "use" computing (C) given that he can "use" geometry (G) (i.e., P(G)=1) is equal to the probability that he can "use" computing.

$$P(C/G) = P(C) \qquad \text{(A5.1)}$$

The information conveyed to the observer of a designer is related to the probability (p_i) of the designer using each of the required notions. For some number of notions N, relating to a specialty b, this information can be expressed for an idea a as:

$$I_{a,b} = \sum_{i}^{N} -\log_2 p_i \qquad \text{(A5.2)}$$

Let the information conveyed to an observer, that is associated with geometry, have a subscript G; and that associated with computing a subscript C. Then, for the subset of notions considered, it can be stated that

$$I_{(\text{Figure } 3.1),G} + I_{(\text{Figure } 3.1),C} > I_{(\text{Figure } 3.2),G} \qquad \text{(A5.3)}$$

because "using" computing and geometrical notions were stated by Proposition A5.1 to be independent.

From equation A5.3 two conclusions can be drawn. First, the inequality is dependent on computing and not geometrical probabilities. Second, unless the designer finds computing notions trivial (their probabilities equal to zero), he will have to convey more information using the representation of Figure 3.1.

APPENDIX 6

COMPUTER OUTPUT FOR EXAMPLE IN SECTION 6.1.1

```
 06-05-68
SIGNED ON AT: 20:55.23
-
HELLO, YOU ARE NOW USING TDS.  DO YOU NEED
INSTRUCTIONS FOR INITIALIZING THIS DESIGN SYSTEM?
PLEASE TYPE "YES" OR "NO".

NO
DEVICE?

338
O.K.
YOU ARE NOW READY TO USE TDS UNDER "338" CONTROL
-
WOULD YOU LIKE INSTRUCTIONS FOR USING TDS, AND A
SUMMARY OF ITS CAPABILITIES? "YES" OR "NO"

NO
O.K.
-

TYPE DESIRED MODE NAME, OR INSERT
MORE INFORMATION INTO PRESENT MODE.

$T/I

 DO YOU NEED INSTRUCTIONS FOR THIS MODE?
PLEASE TYPE YES OR NO.

NO

PLEASE TYPE A 6 CHARACTER NAME
EVAP 7


 PLEASE TYPE A 6 CHARACTER NAME
H1


PLEASE TYPE A 6 CHARACTER NAME
MDOT


PLEASE TYPE A 6 CHARACTER NAME
HTLOAD



PLEASE TYPE A 6 CHARACTER NAME
H2
```

This comment appears every time the NAME or PORTS light button is acttvated.

These labels also appear on the display.

PLEASE TYPE A 6 CHARACTER NAME
COMPR

PLEASE TYPE A 6 CHARACTER NAME
H1

PLEASE TYPE A 6 CHARACTER NAME
MDOT

PLEASE TYPE A 6 CHARACTER NAME
HPWR

PLEASE TYPE A 6 CHARACTER NAME
H2

PLEASE TYPE A 6 CHARACTER NAME
COND 3

PLEASE TYPE A 6 CHARACTER NAME
H1

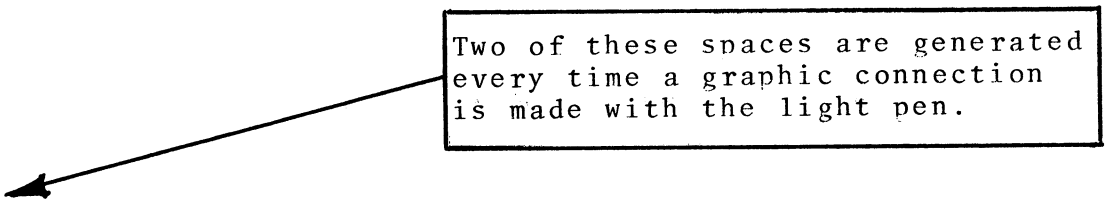
PLEASE TYPE A 6 CHARACTER NAME
DELT

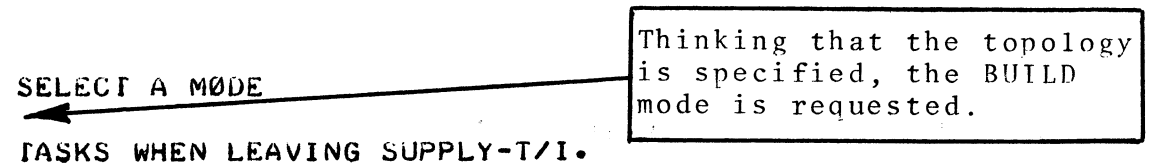
PLEASE TYPE A 6 CHARACTER NAME
OD

PLEASE TYPE A 6 CHARACTER NAME
L

PLEASE TYPE A 6 CHARACTER NAME
HBAR

```
PLEASE TYPE A 6 CHARACTER NAME
H2


PLEASE TYPE A 6 CHARACTER NAME
MDØT


PLEASE TYPE A 6 CHARACTER NAME
VALVE


PLEASE TYPE A 6 CHARACTER NAME
H1


PLEASE TYPE A 6 CHARACTER NAME
H2
```





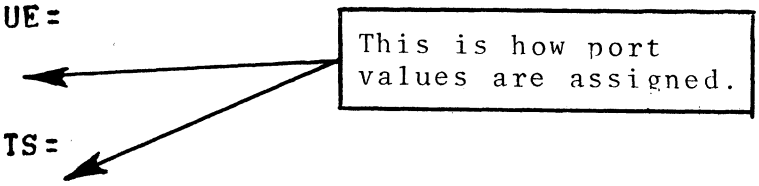

```
PLEASE SELECT A MØDE
$BUILD
  SAVE TASKS WHEN LEAVING SUPPLY-T/I.
DØNE IN T/I


0061
```

```
     IF YOU NEED INSTRUCTIONS ON SELECTING
     THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
     TYPE YES' OTHERWISE NO'.

 THE PRESENT INSTANCE IS " EVAP 7 "

NO

     PLEASE MAKE YOUR SELECTION NOW.

=EVAP1

     IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
     MATCH THE PORTS OF THE INSTANCE"EVAP 7 "
     WITH THE PORTS FOR SUBROUTINE 'EVAP1
     PLEASE TYPE YES' OTHERWISE NO'.

NO

     PLEASE USE THE FOLLOWING
     TO MAKE YOUR MATCHES.

INSTANCE INPUTS: MDOT  ;HTLD  ;H1     /INSTANCE OUTPUTS: H2

SUBROUTINE INPUTS: 059H1;059HTLOAD;059MDOT /SUBROUTINE OUTPUTS: 059H2

MDOT=059MDOT
" INPUT MATCH NO. 1"

H1=059H1
" INPUT MATCH NO. 2"

HTLD=059HTLOAD
" INPUT MATCH NO. 3"

H2=059H2
" OUTPUT MATCH NO. 1"
```

The function of this element function was known by the designer.

A typical "mapping" of an instance port and its corresponding subroutine port.

```
     THE FOLLOWING PORTS FOR "EVAP 7"
     MUST BE GIVEN A VALUE WITH UNITS AT THIS POINT.

PARAMETER " MDOT    " FOR INSTANCE " EVAP 7 "
MATCHED WITH INPUT " 059MDOT " OF SUBROUTINE "EVAP1 "

POSSIBLE UNITS ARE:

LB/MIN LBM/MIN
VALUE=

5

UNITS=

LB/MIN

PARAMETER " HTLD    " FOR INSTANCE " EVAP 7 "
MATCHED WITH INPUT " 059HTLOAD " OF SUBROUTINE "EVAP1 "

POSSIBLE UNITS ARE:

BTU/HR
VALUE=

9600

UNITS=

BTU/HR


TRIAL VALUE " H1      " FOR INSTANCE " EVAP 7  "
MATCHED WITH INPUT " 059H1 " FOR SUBROUTINE " EVAP1

POSSIBLE UNITS ARE:

BTU/LB BTU/LB. BTU./LB BTU./LB.
VALUE=

41
UNITS=

BTU/LB
```



```
011;10062;1

     IF YOU NEED INSTRUCTIONS ON SELECTING
     THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
     TYPE YES' OTHERWISE NO'.

 THE PRESENT INSTANCE IS " COMPR   "

NO

     PLEASE MAKE YOUR SELECTION NOW.

 =FCOMP

     IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
     MATCH THE PORTS OF THE INSTANCE "COMPR   "
     WITH THE PORTS FOR SUBROUTINE "FCOMP    "
     PLEASE TYPE YES' OTHERWISE NO'.

NO

     PLEASE USE THE FOLLOWING
     TO MAKE YOUR MATCHES.

INSTANCE INPUTS: MDOT   ;HPWR   ;H1      /INSTANCE OUTPUTS: H2

SUBROUTINE INPUTS: 056H1;056P;056MDOT /SUBROUTINE OUTPUTS: 0
56H2

MDOT=056MDOT
 " INPUT MATCH NO. 1"

H1=056H1
 " INPUT MATCH NO. 2"

HPWR=056P
 " INPUT MATCH NO. 3"

H2=056H2
   " OUTPUT MATCH NO. 1"
```

```
     THE FOLLOWING PORTS FOR "COMPR "
     MUST BE GIVEN A VALUE WITH UNITS AT THIS POINT.

PARAMETER " MDOT    " FOR INSTANCE " COMPR   "
MATCHED WITH INPUT " 056MDOT " OF SUBROUTINE "FCOMP "

POSSIBLE UNITS ARE:

LB/MIN LBM/MIN
VALUE=

5

UNITS=

LB/MIN

PARAMETER " HPWR    " FOR INSTANCE " COMPR   "
MATCHED WITH INPUT " 056P " OF SUBROUTINE "FCOMP "

POSSIBLE UNITS ARE:

HP HP. HORSEPOWER
VALUE=

-.9

UNITS=

HP




 011;1006;10192;1;3

     IF YOU NEED INSTRUCTIONS ON SELECTING
     THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
     TYPE YES' OTHERWISE NO'.


 THE PRESENT INSTANCE IS " COND 4 "


NO
```

```
     PLEASE MAKE YOUR SELECTION NOW.


=COND1

     IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
     MATCH THE PORTS OF THE INSTANCE "COND 4 "
     WITH THE PORTS FOR SUBROUTINE "COND1   "
     PLEASE TYPE YES' OTHERWISE NO'.

NO

     PLEASE USE THE FOLLOWING
     TO MAKE YOUR MATCHES.


INSTANCE INPUTS: L     ;MDOT   ;HBAR   ;DELT   ;H1     ;OD
INSTANCE OUTPUTS: H2

SUBROUTINE INPUTS: 057H1;057HBAR;057L;057MDOT;0570D;057DELT
/SUBROUTINE OUTPUTS: 057H2

L=057L
" INPUT MATCH NO. 1"


MDOT=057MDOT
" INPUT MATCH NO. 2"


HBAR=057HBAR
" INPUT MATCH NO. 3"


DELT=057DELT
" INPUT MATCH NO. 4"


H1Z_=057H1
" INPUT MATCH NO. 5"


OD=0570D
" INPUT MATCH NO. 6"


H2=057H2
" OUTPUT MATCH NO. 1"
```

```
     THE FOLLOWING PORTS FOR "COND 4"
     MUST BE GIVEN A VALUE WITH UNITS AT THIS POINT.

PARAMETER " L       " FOR INSTANCE " COND 4 "
MATCHED WITH INPUT " 057L " OF SUBROUTINE "COND1 "

POSSIBLE UNITS ARE:

FT FEET FOOT FT.
VALUE=

36

UNITS=

FT

PARAMETER " MDOT    " FOR INSTANCE " COND 4 "
MATCHED WITH INPUT " 057MDOT " OF SUBROUTINE "COND1 "

POSSIBLE UNITS ARE:

LB./HR. LB/HR
VALUE=

5

UNITS=

LB/HR

PARAMETER " HBAR    " FOR INSTANCE " COND 4 "
MATCHED WITH INPUT " 057HBAR " OF SUBROUTINE "COND1 "

POSSIBLE UNITS ARE:

BTU/FT^2-F-HR
VALUE=

45

UNITS=

BTU/FT^2-F-HR

PARAMETER " DELT    " FOR INSTANCE " COND 4 "
MATCHED WITH INPUT " 057DELT " OF SUBROUTINE "COND1 "
```

```
POSSIBLE UNITS ARE:

F FDEG
VALUE=

65

UNITS=

F

PARAMETER " OD      " FOR INSTANCE " COND 4 "
MATCHED WITH INPUT " 0570D " OF SUBROUTINE "COND1 "

POSSIBLE UNITS ARE:

IN IN. INCHES
VALUE=

.25

UNITS=

IN

     IF YOU NEED INSTRUCTIONS ON SELECTING
     THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
     TYPE YES' OTHERWISE NO'.

 THE PRESENT INSTANCE IS " VALVE   "

NO

     PLEASE MAKE YOUR SELECTION NOW.

=VALV1

     IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
     MATCH THE PORTS OF THE INSTANCE "VALVE   "
     WITH THE PORTS FOR SUBROUTINE "VALV1    "
     PLEASE TYPE YES' OTHERWISE NO'.

NO

     PLEASE USE THE FOLLOWING
     TO MAKE YOUR MATCHES.
```

```
INSTANCE INPUTS: H1      /INSTANCE OUTPUTS: H2

SUBROUTINE INPUTS: 061H1 /SUBROUTINE OUTPUTS: 061H2

H1=061H1
" INPUT MATCH NO. 1"


H2=061H2
" OUTPUT MATCH NO. 1"

ALL DONE IN BUILD

$EXECUTE

 AN ENTRY INTO EXECUTE MUST BE CONFIRMED.
PLEASE CONFIRM OR CANCEL.

OK

 001
 SUBROUTINE " EVAP1 " WILL NOW BE PROCESSED.
... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5059,5059.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  106246
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     7:50 PM    JUNE 4, 1968
```

TDS recognized that all information needed to analyze the problem has been received.

The designer wanted some answers.

The first element function is now to be processed. The '#' signs are printed when the central computer is operating automatically from a file of commands which were written by TDS.

```
  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...


  006
  SUBROUTINE " FCOMP " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5056,5056.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10624E
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     7:51 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...
  011
  SUBROUTINE " COND1 " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5057,5057.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10628A
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS
```

```
  (MOD: F-03/02/03)     7:53 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...

  019
  SUBROUTINE " VALV1 " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5061,5061.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10621A
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     7:55 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...


EXECUTE 2.105

EXECUTE 2.105
FOLLOWING ARE THE VALUES
PRODUCED BY YOUR SYSTEM.
```

This is the answer. Note that (1) 41.0 BTU/LB was the trial value, and (2) all four programs were individually executed and INTER-program formatting was AUTOMATIC---all in 4+ minutes.

```
 PORT " H2      " FOR INSTANCE " VALVE    "
HAS THE VALUE "57.67374 BTU/LB"
THOSE ARE ALL THE OUTSTANDING VALUES.
```

```
TYPE DESIRED MODE NAME, OR INSERT
MORE INFORMATION INTO PRESENT MODE.

$T


 THIS IS T/I


 IF YOU WOULD LIKE INSTRUCTIONS
FOR USING THIS MODE PLEASE TYPE
"YES" OTHERWISE "NO".

NO

PLEASE TYPE DESIRED OPERATOR NAME

$ALT/057DELT:65;140/
NEW VALUE OF "057DELT" IS "140".

$EXECUTE
  SAVE TASKS WHEN LEAVING SUPPLY-T/I.




 AN ENTRY INTO EXECUTE MUST BE CONFIRMED.
PLEASE CONFIRM OR CANCEL.

OK

  001
  SUBROUTINE " EVAP1 " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5059,5059.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
```

The temperature drop in the condenser is being changed so that the answer from the next execution will more closely approach the trial value.

```
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  106246
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     9:31 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...


  006
  SUBROUTINE " FCOMP " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5056,5056.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10624E
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     9:32 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...
```

```
  011
  SUBROUTINE " COND1 " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5057,5057.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10628A
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     9:35 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...

$QUE/
$UMI/VALUE/


31.18002


$ALT/057DELT:140;115/
NEW VALUE OF "057DELT" IS "115".

$EXECUTE
```

This value should correspond to the trial value of 41.0.

The temperature drop will now be reduced and an execution tried.

```
  AN ENTRY INTO EXECUTE MUST BE CONFIRMED.
 PLEASE CONFIRM OR CANCEL.

 OK


   001
   SUBROUTINE " EVAP1 " WILL NOW BE PROCESSED.
  ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5059,5059.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP       0  ***** RESTART AT LOCATION  106246
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


   (MOD: F-03/02/03)      9:43 PM    JUNE 4, 1968

   <> <> <> TRAMP, ASSOCIATIVE VERSION OF
  ... UMIST SIGNS ON ...


   006
   SUBROUTINE " FCOMP " WILL NOW BE PROCESSED.
  ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5056,5056.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
```

```
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10624E
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     9:45 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...
  011
  SUBROUTINE " COND1 " WILL NOW BE PROCESSED.
 ... UMIST SIGNS OFF ...
#EXECUTION TERMINATED
#$SOURCE BUILD(150)
#$SOURCE EXECUTE(150)
#$COPY LIB(5057,5057.999) TO -YUP
#$RUN *DUMMY*
#ATTEMPT TO LOAD A NULL PROGRAM.
#$EMP -OUTPUT
#DONE.
#$RUN -YUP 1=-INPUT 2=-OUTPUT
#EXECUTION BEGINS
 IHC002I STOP      0  ***** RESTART AT LOCATION  10628A
#EXECUTION TERMINATED
#$EMP -INPUT
#DONE.
#$EMP -YUP
#DONE.
#$RUN TRAMP SCARDS=EXECUTE(2.01) PAR=P
#EXECUTION BEGINS


  (MOD: F-03/02/03)     9:46 PM    JUNE 4, 1968

  <> <> <> TRAMP, ASSOCIATIVE VERSION OF
 ... UMIST SIGNS ON ...


 $UMI/VALUE/


 40.01128
```

The three passes enabled the parameters to be adjusted so that the trial value of 41.0 was reasonably approached.

APPENDIX 7

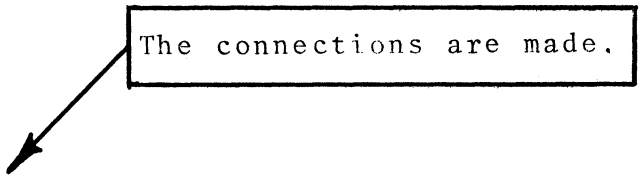
COMPUTER OUTPUT FOR EXAMPLE IN SECTION 6.1.2

```
 06-05-68
SIGNED ON AT: 14:30.55


HELLO, YOU ARE NOW USING TDS.  DO YOU NEED
INSTRUCTIONS FOR INITIALIZING THIS DESIGN SYSTEM?
PLEASE TYPE "YES" OR "NO".

NO
DEVICE?

TTY
O.K.
YOU ARE NOW READY TO USE TDS UNDER TELETYPE CONTROL


WOULD YOU LIKE INSTRUCTIONS FOR USING TDS, AND A
SUMMARY OF ITS CAPABILITIES? "YES" OR "NO"

NO
O.K.



TYPE DESIRED MODE NAME, OR INSERT
MORE INFORMATION INTO PRESENT MODE.

$T


 THIS IS T/I


 IF YOU WOULD LIKE INSTRUCTIONS
FOR USING THIS MODE PLEASE TYPE
"YES" OTHERWISE "NO".

NO

PLEASE TYPE DESIRED OPERATOR NAME

%I/LOAD/

 INSTANCE "LOAD   " IS NOW IN THE "MENU"

%I/MOMI/
```

In this example, the topology is inputted in the teletype [TTY] mode.

```
 INSTANCE "MOMI   " IS NOW IN THE "MENU"
%I/DIV 1/

 INSTANCE "DIV 1  " IS NOW IN THE "MENU"
%I/DIV 2/

 INSTANCE "DIV 2  " IS NOW IN THE "MENU"
%I/BATT/

 INSTANCE "BATT   " IS NOW IN THE "MENU"
%I/GAGE 1/

 INSTANCE "GAGE 1" IS NOW IN THE "MENU"
%I/GAGE 2/

 INSTANCE "GAGE 2" IS NOW IN THE "MENU"
%I/SCOPE/

 INSTANCE "SCOPE  " IS NOW IN THE "MENU"
%P/LOAD//P:DR1;P:DR2;P:DR3;P:DR4/
%P/LOAD//P:R;P:F;P:G;P:XL;P:E/
%P/LOAD//I:C;I:XI;O:R1;O:R2;O:R3;O:
%P/MOMI//T:B;P:H;O:C;O:XI/
%P/DIV 1//O:R11;I:R1;O:R12/
%P/DIV 2//O:R31;I:R3;O:R32/
%P/BATT//I:R12;I:R2;I:R4;I:R32/
%P/BATT//P:EBB;O:XIL;O:XIR/
%P/GAGE 1//P:EBB;I:R11;I:XIL;O:EX/
```

```
%P/GAGE 2//P:EBB;I:R31;I:XIR;O:EY/
%P/SCOPE//I:EX;I:EY;O:EO/


%C/LOAD/
%C/MOMI/
%C/DIV 1/
%C/DIV 2/
%C/BATT/
%C/GAGE 1/
%C/GAGE 2/
%C/SCOPE/
%M/MOMI:C//LOAD:C//1000/
%M/MOMI:XI//LOAD:XI//1010/
%M/MO
%M/LOAD:R1//DIV 1:R1//1020/
%M/LOAD:R3//DIV 2:R3//1030/
%M/LOAD:R2//BATT:R2//1040/
%M/LOAD:R4//BATT:R4//1050/
%M/DIV 1:R12//BATT:R12//1060/
%M/DIV 2:R32//BATT:R32//1070/
%M/BATT:XIL//GAGE 1:XIL//1080/
%M/BATT:XIR//GAGE 2:XIR//1:_090/
%M/GAGE 1:EX//SCOPE:EX//1100/
%M/GAGE 2:EY//SCOPE:EY//1200/
```



```
%M/DIV 1:R11//GAGE 1:R11//1300/

%M/DIV 2:R31//GAGE 2:R31//1400/

%
$BUILD#(SAVE,-HOPE1)
  SAVE TASKS WHEN LEAVING SUPPLY-T/I.
DONE IN T/I




 0011

      IF YOU NEED INSTRUCTIONS ON SELECTING
      THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
      TYPE YES' OTHERWISE NO'.


 THE PRESENT INSTANCE IS " MOMI    "


NO

      PLEASE MAKE YOUR SELECTION NOW.


=MOMI

      IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
      MATCH THE PORTS OF THE INSTANCE"MOMI    "
      WITH THE PORTS FOR SUBROUTINE "MOMI    "
      PLEASE TYPE YES' OTHERWISE NO'.

NO

      PLEASE USE THE FOLLOWING
      TO MAKE YOUR MATCHES.


INSTANCE INPUTS: H      ;B       /INSTANCE OUTPUTS: XI     ;C

SUBROUTINE INPUTS: 071B;071H /SUBROUTINE OUTPUTS: 071C;071XI

H=071H
" INPUT MATCH NO. 1"
```

```
B =071B
"  INPUT MATCH NO. 2"


XI =071XI
"  OUTPUT MATCH NO. 1"

C =071C
"  OUTPUT MATCH NO. 2"




     THE FOLLOWING PORTS FOR "MOMI   "
     MUST BE GIVEN A VALUE WITH UNITS AT THIS POINT.

PARAMETER " H       " FOR INSTANCE " MOMI    "
MATCHED WITH INPUT " 071H " OF SUBROUTINE "MOMI  "

POSSIBLE UNITS ARE:

IN IN. INCHES
VALUE=

.25

UNITS=

IN


TRIAL VALUE " B       " FOR INSTANCE " MOMI    "
MATCHED WITH INPUT " 071B " FOR SUBROUTINE " MOMI

POSSIBLE UNITS ARE:

IN IN. INCHES
VALUE=

1.5
UNITS=

IN
```

```
001;10051;2



001;1004;10051;3;2



001;1004;1003;10051;3;4;2

     IF YOU NEED INSTRUCTIONS ON SELECTING
     THE PROPER SUBROUTINE IN THE LIBRARY PLEASE
     TYPE YES' OTHERWISE NO'.

 THE PRESENT INSTANCE IS " LOAD    "

NO

     PLEASE MAKE YOUR SELECTION NOW.

=LOAD

     IF YOU WOULD LIKE INSTRUCTIONS ON HOW TO
     MATCH THE PORTS OF THE INSTANCE "LOAD   "
     WITH THE PORTS FOR SUBROUTINE 'LOAD    '
     PLEASE TYPE YES' OTHERWISE NO'.

NO

     PLEASE USE THE FOLLOWING
     TO MAKE YOUR MATCHES.

INSTANCE INPUTS: G      ;DR4    ;DR3    ;DR2    ;DR1    ;XL     ;R
     ;E      ;XI     ;F      ;C       /INSTANCE OUTPUTS: R4     ;
R3     ;R2     ;R1

SUBROUTINE INPUTS: 064F;064R;064G;064XL;064C;064E;064XI;064D
R1;064DR2;064DR3;064DR4 /SUBROUTINE OUTPUTS: 064R1;064R2;064
R3;064R4
```

```
G=064G
"  INPUT MATCH NO. 1"

DRI_1=064DR1
"  INPUT MATCH NO. 2"

DR2=064DR2
"  INPUT MATCH NO. 3"

DR3=064DR3
"  INPUT MATCH NO. 4"

DR4=064DR4
"  INPUT MATCH NO. 5"

XL=064XL
"  INPUT MATCH NO. 6"

R=064R
"  INPUT MATCH NO. 7"

E=064E
"  INPUT MATCH NO. 8"

XI=064XI
"  INPUT MATCH NO. 9"

F=064F
"  INPUT MATCH NO. 10"

C=064C
"  INPUT MATCH NO. 11"

R1=064R1
"  OUTPUT MATCH NO. 1"

R2=064R2
"  OUTPUT MATCH NO. 2"

R3=064R3
"  OUTPUT MATCH NO. 3"

R4=064R4
#OFF AT 16:16.59
```

The disc being used in the central computer became "locked out" for some reason, and the problem was terminated by the machine operator.

APPENDIX 8

EXAMPLE OF ENTERING AN ANALYSIS PROGRAM INTO TDS' LIBRARY

```
TYPE DESIRED MØDE NAME, ØR INSERT
MØRE INFØRMATIØN INTØ PRESENT MØDE.

$LIB
THIS IS LIBRARY.
PLEASE SELECT WHICH ØF THE FØLLØWING MØDES IN THE LIBRARY
YØU WISH TØ USE:
     (1)     ESTABLISH
     (2)     ALTER
     (3)     DESTRØY
     (4)     YØUR ØWN  ADDITIØNS TØ THE LIBRARY

%EST
THIS IS ESTABLISH.
PLEASE TYPE THE NAME ØF THE SUBRØUTINE YØU WISH TØ ENTER.

G2
PLEASE TYPE EACH INPUT PØRT FØR SUBRØUTINE"G2"
AND ITS NEEDED UNITS IN THE FØLLØWING FØRMAT:
     "PØRT NAME - UNITS"
EXAMPLE XK5J9 - LB./SQ.IN.
 WHEN ALL INPUT PØRTS HAVE BEEN SPECIFIED PLEASE TYPE "**".

XIR - AMPS

EBB - VØLTS

R31 - ØHMS

**
PLEASE TYPE EACH ØUTPUT PØRT FØR SUBRØUTINE"G2" AND
ITS NEEDED UNITS IN THE SAME FØRMAT AS FØR THE INPUT PØRTS.
WHEN ALL ØUTPUTS HAVE BEEN SPECIFIED PLEASE TYPE "**".

EY - VØLTS

**
NØW YØU MUST GIVE THE INPUT AND ØUTPUT FØRMAT FØR DATA
NEEDED AND PRØDUCED BY "G2".
IF YØU NEED INSTRUCTIØNS ØN ENTERING FØRMAT DESCRIPTØS
PLEASE TYPE YES' ØTHERWISE NØ'.

NØ
THE FØRM IS PØRT NAME-STARTING CØLUMN-FIELD LENGTH-MØDE

 * * * * LINE " 1 " * * * *

XIR-1-10-F

EB B-11-10-F

R31-21-10-F
```

```
**

 * * * * LINE " 2 " * * * *

*
ØUTPUT FØRMATTING IS EXACTLY THE SAME.  PLEASE
USE THE SAME FØRM ØF FØRMAT DESCRIPTØR.

 * * * * LINE "1" * * * *

EY-1-10-F

**

 * * * * LINE "2" * * * *

*

  * * * * * * * * INPUTS * * * * * * * * *

 PØRT NAME      LINE NØ.  STARTING CØLUMN FIELD LENGTH  MØDE

 67XIR          1           1         10         FLØATING PØINT          AM
S

 67EBB          1           11         10         FLØATING PØINT          V
LTS

 67R31          1           21         10         FLØATING PØINT          Ø
MS


 * * * * * * * * ØUTPUTS * * * * * * * * *

 PØRT NAME      LINE NØ.  STARTING CØLUMN FIELD LENGTH  MØDE

 67EY          1           1         10         FLØATING PØINT          V@L
S



 IS EVERYTHING ABØVE ACCEPTABLE? "YES" ØR "NØ"

YES
```

```
 IS THE PRØGRAM TØ BE LØADED IN SØURCE ØR ØBJECT CØDE?

SØURCE


 WHERE MAY THE SØURCE DECK BE FØUND?

-W4

YØUR PRØGRAM WILL NØW BE LØADED
 ••• UMIST SIGNS ØFF •••
#EXECUTIØN TERMINATED
#$SØURCE BUILD(150)
#$SØURCE LIB(10000)
#$RUN *FØRTRAN SCARDS=-W4 SPRINT=*DUMMY* SPUNCH=LIB(5067,10000,.01)
#EXECUTIØN BEGINS
#EXECUTIØN TERMINATED
#$CØPY -W4 TØ LIB(7067,10000,.01)
#$RUN TRAMP SCARDS=LIB(199.35) SPRINT=*SINK* PAR=P
#EXECUTIØN BEGINS


 (MØD: F-03/02/03)    10:15 AM    JUNE 5, 1968

 <> <> <> TRAMP, ASSØCIATIVE VERSIØN ØF
••• UMIST SIGNS ØN •••
```

BIBLIOGRAPHY

Systems Engineering

Affel, Herman A., Jr. "Systems Engineering," International Science and Technology, November 1964.

Dixon, John R. Design Engineering: Inventiveness, Analysis and Decision Making, McGraw-Hill, New York, 1966.

Eder, W.E. Mechanical System Design, Pergamon, New York, 1965.

Esherick, Joseph. "Problems of the Design of a Design System," Conference on Design Methods, Pergamon, New York, 1963.

Feigenbaum, D.S. "Systems Engineering: A Major New Technology," Industrial Quality Control, September 1963.

Hall, A.D. A Methodology for Systems Engineering, Van Nostrand, Princeton, N.J., 1962.

Kuo, F.F., and J.F. Kaiser. System Analysis by Digital Computer, Wiley, New York, 1966.

Lind, N.C. "Analysis of Structures by Systems Theory," Journal of the Structural Division - A.S.C.E., Vol. 88, April 1962.

Machol, R.E. Systems Engineering Handbook, McGraw-Hill, New York, 1965.

Mesarovic, M.D., Editor. Second Systems Symposium, Case Institute of Technology, Wiley, New York, 1964.

Motard, R.L. "Systems Engineering: Engineering Come of Age," Journal - A.S.C.E., Vol. 92, 1966.

Paynter, Henry M. Analysis and Design of Engineering Systems, M.I.T. Press, Cambridge, Mass., 1960.

Schoenfeld, J.C. "Analog of Hydraulic, Mechanical, Acoustic and Electrical Systems," Applied Scientific Research, Section B, Vol. #, No. 6, 1954.

Wilson, W. Concepts of Engineering System Design, McGraw-Hill, New York, 1965.

Networks, Graphs and Their Matrices

Berge, Claude, and A. Ghouila-Houri. Programming, Games and Transportation Networks, Wiley, New York, 1965.

Busacker, Robert G. and Thomas L. Saaty. Finite Graphs and Networks: an Introduction with Applications, McGraw-Hill, New York, 1965.

Cederbaum, I. "Application of Matrix Algebra to Network Theory," I.R.E. Transactions, Vol. CT-6, May 1959.

Darlington, S. "A Survey of Network Realization Techniques," I.R.E. Transactions, Vol. CT-2, February 1955.

Fenves, S.J. and F.H. Branin Jr. "A Network Topological Formulation of Structural Analysis," IBM Technical Report No. 00.979-1, September 22, 1964.

Ford, L.R. and D.R. Fulkerson. Flows in Networks, Princeton University Press, Princeton, N.J., 1962.

Gordon, C.K. "The Mathematics of a Structure," International Science and Technology, No. 53, May 1966.

Gould, R. "Graphs and Vector Spaces," Journal of Mathematical Physics, Vol. 37, 1958.

Grossman, Israel and Wilhelm Maguus. Groups and Their Graphs, Random House, New York, 1964.

Harary, Frank, Robert Z. Norman, and Darwin Cartwright. Structural Models : An Introduction to the Theory of Directed Graphs, Wiley, New York, 1965.

Huelsman, R.G. Circuits, Matrices and Linear Vector Spaces, McGraw-Hill, New York, 1963.

Kim, W.H. and R.T. Chien. Topological Analysis and Synthesis of Communication Networks, Columbia University Press, New York, 1962.

Marcus, M. Basic Theorems in Matrix Theory, U.S. Government Printing Office, Washington, D.C., 1960.

Mowry, J.W. "Problem Organization Using Linear Transforms," A term paper for course ME 642, The University of Michigan, Ann Arbor, April 1966.

Mowshowitz, Abbe. Entropy and the Complexity of Graphs, Concomp Project Technical Report, The University of Michigan, Ann Arbor, August 1967.

Norman, R.L. "A Matrix Method for Location of Cycles of a Directed Graph," A.I.Ch.E. Journal, May 1965.

Ore, Oystein. Graphs and Their Uses, Random House, New York, 1963.

Pullen, K.A. Theory and Application of Topological and Matrix Methods, Rider, New York, 1960.

Reiter, Raymond. A Study of a Model for Parallel Computations, Ph.D. Dissertation, Systems Engineering Laboratory, The University of Michigan, Ann Arbor, June 1967.

Sanford, R.S. Physical Networks, Prentice-Hall, Englewood Cliffs, N.J., 1965.

Seshu, S. and M.B. Reed. Linear Graphs and Electrical Networks, Addison-Wesley, Reading, Pa., 1961.

Shearer, J.L., A.T. Murphy, H.H. Richardson. Dynamic Systems, Vol. I and II, Pre-Publication Edition, Addison-Wesley, Reading, Pa., 1965.

Wineberg, L. Network Analysis and Synthesis, McGraw-Hill, New York, 1962.

Zadeh, L. Linear System Theory: The State-Space Approach, McGraw-Hill, New York, 1963.

Information Theory

Attneave, Fred. Applications of Information Theory to Psychology, Holt, Rinehart and Winston, New York, 1959.

Bell, David A. Information Theory and Its Engineering Applications, Pitman, London, 1962.

Goldman, Stanford. Information Theory, Prentice-Hall, Englewood Cliffs, N.J., 1953.

Proceedings of the Symposium on Information Networks, Vol. III, Polytechnic Institute of Brooklyn Press, New York, April 1954.

Sass, Margo A. and William D. Wilkinson. Computer Augmentation of Human Reasoning, Spartan Books, New York, 1965.

Shannon, Claude E. and Warren Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana, 1949.

Tou, Julius T. and Richard H. Wolcox. Computer and Information Sciences, Spartan Books, New York, 1964.

Whitehill, Joseph. "Reappraisals: II - Samuel Taylor Coleridge: Prisoner and Prophet of System," The American Scholar, Vol. 37, No. 1, Winter 67-68.

Wilson, Ira G. and Marthann E. Wilson. Information Computers and System Design, Wiley, New York, 1965.

Computing - General

Ash, W.L. and E.H. Sibley. TRAMP: A Relational Memory with an Associative Base, Technical Report, Concomp Project, University of Michigan, Ann Arbor, June 1968.

Bellman, R. "Dynamic Programming," Science, Vol. 153, No. 3731, Namuary 7, 1966.

Bellman, R. and R. Karusy. Dynamic Programming: A Bibliography of Theory and Application, RAND Corp, Santa Monica, Calif., August 1964.

Blake, K. and G. Gordon. "Systems Simulation with Digital Computers," IBM Systems Journal, Vol. 3, No. 1, 1964.

Campbell, H.G. An Introduction to Matrices, Vectors and Linear Programming, Appliton-Century-Crofts, New York, 1965.

Childs, David L. Description of a Set Theoretic Data Structure, Technical Report No. 3, Concomp Project, University of Michigan, Ann Arbor, March 1968.

Galler, B.A. and A.J. Perlis, "A Proposal for Definitions in ALGOL," Communications of the ACM, Vol. 10, No. 4, April 1967.

Mooers, C.N. "TRAC, A Procedure-Describing Language for the Reactive Typewriter," Communications of the ACM, Vol. 9, No. 3, March 1966.

Mooers, C.N. and L.P. Deutsch, "TRAC, A Text-Handling Language," ACM 20th Anniversary, Proceedings,1965.

Murphy, J.S. Basics of Digital Computer Programming, Rider, New York, 1964.

MTS: Michigan Terminal System, University of Michigan Computing center, Ann Arbor, 1968.

Sibley, E.H. The Computer as Symbol Manipulator and Evaluator for Engineering, A Ph.D. Dissertation at the Massachusetts Institute of Technology, Cambridge, 1967.

Tucker, Stephen G. "Automated Program Documentation," TRW Summer Student Assistant Program in Computer Science, Section 15, TRW Systems, Redondo Beach, Calif., 1967.

Computer-Aided Design

Chasen, S.H. "Man-Computer Graphic Communication," Machine Design, Penton Publications, Cleveland, Ohio, March 3, 1966.

Ling, Marvin T. The Logical and Analytical Structure of the Computer-Aided Design Process as Applied to a Class of Mechanical Design Problems, A Ph.D. Dissertation at the University of Michigan, Ann Arbor, 1964.

Roos, D. ICES System Design, M.I.T. Press, Cambridge, 1966.

Rosenberg, Ronald C. Computer-Aided Teaching of Dynamic System Behavior, A Ph.D. Dissertation at Massachusetts Institute of Technology, Cambridge, 1965.

Sutherland, W.R. The On-Line Graphical Specification of Computer Procedures, A Ph.D. Dissertation at the Massachusetts Institute of Technology, Cambridge, 1966.

Westervelt, F.H. A Study of Automatic System Simulation Programming and the Analysis of the Behavior of Physical Systems Using an Internally Stored Program Computer, A Ph.D. Dissertation at The University of Michigan, Ann Arbor, 1960.

Simulation

Harman, H.H. "Simulation as a Tool for Research," System Development Corporation, Santa Monica, Cal., SP-565, 1961.

Harman, H.H. "The System Simulation Research Laboratory," System Development Corporation, Santa Monica, Calif., TM-498, 1960.

Mitchell, R.K. and F.A. Creswick. "The Synthesis and Use of Mathematical Models as Aids to Design," ASME Paper 64-MD-45, 1964.

Ripperger, W.C. MIMIC: A Digital Simulator Program, University of Michigan Computing Center, Ann Arbor, 1966.

Shigley, Joseph E. Simulation of Mechanical Systems, McGraw-Hill, New York, 1967.