ON-LINE COMPUTING SYSTEMS

The University of California at Los Angeles
and
Informatics Inc.

ON-LINE COMPUTING SYSTEMS

EDITED BY
ERIC BURGESS

DATA PROCESSING
LIBRARY SERIES

Proceedings of the Symposium sponsored by
The University of California
at Los Angeles and Informatics Inc.

February 2-4, 1965
Los Angeles, California

22nd Floor, Book Tower, Detroit, Michigan 48226

Foreword

EARLY IN ITS HISTORY as a corporate entity, Informatics Inc. sponsored a symposium on disc files. The response to that symposium was beyond all expectations. More applicants were turned away than were able to register.

Experience with that disc file symposium showed that much management time and effort must be devoted to the staging and programming of a successful presentation. Therefore, Informatics Inc. proposed to Engineering Extension of the University of California, Los Angeles, joint work on an On-Line Computing Systems Symposium to be held early in 1965. Acceptance of this proposal by UCLA meant that the full facilities, prestige, and know-how of a great university could be brought to work on this symposium. The results have fully vindicated the wisdom of this decision.

The final registration of 768 shows the great current interest in on-line systems. This registration also exceeded all estimates.

The cooperation of Informatics Inc. with its heavy, first-hand skill and interest in programming and specifying on-line systems, and the University of California, devoted through such organizations as Engineering Extension to bringing new and current fields of knowledge to people, provided an outstanding example of the benefits to be gained by education and industry working together.

Jackson W. Granholm
Sherman Oaks, California

Library of Congress Catalog Card Number: 65-21221

Preface

ON-LINE DATA PROCESSING SYSTEMS have recently become of interest in digital computer applications. Developments in digital transmission and availability of faster bulk storage devices and the use of man/machine interface devices have stimulated a new kind of data processing. In this processing, information is entered into the system as it is generated. Outputs are requested as they are required. These inputs and outputs are occasioned by external stimuli—man or machine—to which the computer responds.

On-line computing systems include at least two important classes of systems. The first is one in which response times are measured in milliseconds. Such systems are automatic, and many of them are closed loop, since the timing requirements preclude the intervention of men. Examples are process control applications, military satellite control systems, and radar tracking and recording systems.

The second important class includes computer systems to which several interrogation and display devices are connected, thus establishing man/machine communication. Examples are found in military command and control systems, space vehicle command and control systems, and various commercial systems.

The three-day symposium at the University of California Extension, Los Angeles, sponsored by the Department of Engineering and Informatics Inc. included discussion of both classes of on-line systems. In addition, it covered, with a considerable degree of thoroughness, the principles, disciplines, and practices which are applicable to on-line systems design, both in machinery and programming.

The symposium was divided into six morning and afternoon sessions each with a separate chairman. These sessions and their chairmen were:

Session I: Motivations
Chairman—Dr. Gerald Estrin, Professor of Engineering, UCLA

Session II: Techniques
Chairman—Francis V. Wagner, Vice President, Plans and Programs, Informatics Inc.

Session III: Approaches
Chairman—Dr. Michel A. Melkanoff, Associate Professor of Engineering, UCLA

Session IV: Methods
Chairman—Jackson W. Granholm, Vice President, Technical Communications, Informatics Inc.

Session V: Applications
Chairman—Dr. Bertram Bussell, Assistant Professor of Engineering, UCLA

Session VI: Examples and Summary
Chairman—Irving Cohen, Vice President, Command and Control, Informatics Inc.

The proceedings are organized in parts to correspond with the sessions.

The Welcome Address was given by Dr. Paul H. Sheats, Dean, University of California Extension. The Banquet Address was by Dr. Simon Ramo, Vice Chairman of the Board, Thompson Ramo Wooldridge Inc., and President, Bunker-Ramo Corporation. Dr. Walter F. Bauer, President, Informatics Inc., was Chairman of the Banquet, and Jackson W. Granholm, Vice President, Technical Communications, Informatics Inc., was Toastmaster.

The papers for the symposium were selected by an advisory board consisting of:

Dr. G. Estrin, Dr. C. B. Tompkins and Dr. S. Houston, of UCLA, and Dr. W. F. Bauer, F. V. Wagner, and J. W. Granholm, of Informatics Inc.

Secretarial assistance was given by Mrs. Betty Leventhal of UCLA and Mrs Rose Marie Gonzales of Informatics Inc. Public Relations were handled by Tom Kramer of UCLA, and Frank Crane and Robert Stone of Informatics Inc.

The assistance of many other people at UCLA and Informatics Inc. who helped to make the symposium possible is also gratefully acknowledged.

Eric Burgess
Editor,
Informatics Inc.

CONTENTS

PART IV—METHODS

PART V—APPLICATIONS

PART VI—EXAMPLES AND SUMMARY

I

MOTIVATIONS

Dr. Ivan E. Sutherland*

The Future of On-Line Systems

THE SYSTEMS

THERE ARE SIX KINDS of on-line systems:

1. *Systems for Processing Control* let factories manufacture products more cheaply. Process control systems do everything from simple feedback control to optimizing profits through linear programming. These process control systems have, and will have, a big effect on our domestic production capability.
2. *Inqury Systems* permit people at many different locations to find out what is going on. Most familiar is the airline reservations system, but more such systems will come into use in the years ahead.
3. *Specialized On-Line Systems* perform particular complicated tasks; often military tasks. Industry is only just beginning to make use of specialized on-line systems for engineering and design.
4. *On-Line Programming Systems* put the raw power of a computer at the immediate disposal of a human user. Evidence of today's great interest in on-line programming systems is that more and more of them are being used.
5. *On-Line Problem-Solving Systems* will be required for doing even the simplest tasks without human help. Such systems will require the techniques for pattern recognition, process control, and heuristic programming, and will unite them meaningfully. It will be so difficult to do even the simplest tasks automatically that we will be busy with these tasks for some time to come.
6. *On-Line Instrumentation* will bring us better understanding of the interplay of the programs and data within the computer. Simple devices and programs to keep track, on-line, of what the computer does will bring us better understanding of what our information reprocessing systems are actually doing.

THE FUTURE

I will try to divide the future into two categories: the *immediate* future and the *distant* future.

During the immediate future, we can expect systems to come to fruition which we now know how to design and how to build. In talking about systems for the immediate future, we talk about systems which have some recognizable place in the current scheme of society. In the immediate future, there probably will not be any tremendous political upheaval or war, or natural catastrophe which would cause the entire technology of the world to take a new turn and, thus, render my predictions ridiculous.

But, the future is long—probably longer than the past. Information processing is something we have proved can be done, and we are going to do more of it. I believe that, in principle, it is possible to make information processing systems which will do intellectual tasks that human beings cannot possibly hope to do. The creation of such systems is a challenge that society *will* accept. The only question is when? About when we can only speculate, because the future of any technology is so interwoven with the political, social, and scientific developments around it.

SYSTEMS OF THE FUTURE

1. Process Control Systems—There is no reason why Man should have to work for a living. Everyone recognizes the trend toward more and more leisure time—time in which our

*Director for Information Processing Techniques
Advanced Research Projects Agency
Department of Defense
The Pentagon
Washington, D. C.

activities are not prescribed. Leisure time is not necessarily idle time; if we do nothing during leisure time the world will continue as before. In the foreseeable future, process control and automation will make possible more leisure time for all of us.

One of the major social issues yet to be faced is how to measure an individual's contribution in a leisure society. Today, we pay people for working, or for concentrating on a single assembly operation for a long and unpleasant period, or for being away from home, or for taking personal risk, or for artistry—for making or doing something "beautiful", or for inventiveness—for devising something which makes life more pleasant for other people. Or, we pay people for responsibility—for making decisions which will have to stand the test of history and expose the decider to historical recognition or historical contempt. In a leisure society, what would people be paid for? How would you recognize a person's contribution to society? If no one works, except when he wants to, or no one spends long onerous hours at menial labor, except if he wants to, how do we recognize a man's contribution to society? The long term future of on-line systems for process control and automation rests in our ability to answer these questions.

2. Inquiry Systems—Today, you can find out from anywhere in the country what space is available on any airplane, almost instantly. But that is only the beginning. In the foreseeable future, we could automate all sorts of information retrieval, from isolated inventories to the entire content of the Library of Congress.

An important part of an information retrieval system is its completeness. If I could reach 75 per cent of the technical literature and 99 per cent of the technical experts in a field through a certain information retrieval system, I would not need to keep a personal library. Finding out what had been done in a certain field would be a simple one-inquiry task. Just as the utility of the telephone system is that everyone has a telephone, so the full potentiality of an information retrieval system will come only when it contains all pertinent information and almost everyone interested uses it.

Today, our inquiry systems are systems of which we ask questions, and not systems which can phrase and ask questions of human beings. On-lineness is a two-way street, and our success in using it comes from our willingness to make systems which can ask us questions as well as give replies to questions asked. Think of Socrates teaching by merely asking questions!

Automated libraries will be most useful when they "understand" the information stored in them. Suppose you wanted to find out some fairly obscure technical fact. You could go to the library and ask the sweet young librarian a technical question. The librarian may know all about the books in the library and where they are stored, and what the numbering systems mean, and the procedures for signing them out, but she has not the foggiest notion of the *content* of all these books—and certainly not of the book which will contain the information you want.

But imagine asking a technical colleague the same question. Your colleague has at his command the content of the book which contains the facts you need. He knows nothing about libraries, or numbering systems, or information retrieval, or cataloging methods, but he does know the facts that you want. Inquiry systems, in the future, will become more and more able to understand and correlate the facts.

Imagine, if you will, in the far, far, distant future a computer which contains in its files all that has ever been written. In its spare time, this computer mulls over these facts to understand the implications of them and to try to come to new conclusions. This computer knows, of course, the interests of all the people it serves, because it has records of all the questions they have ever asked it. Given a new question, this machine of the future can not only regurgitate the information which it contains but also can put the asker in communication with other people interested in that subject.

3. Elaborate Special Purpose On-Line Systems have been devised primarily for military purposes. In the immediate future, we can expect industry to start using specialized on-line systems for design and management. There are obvious benefits to automatically performing mathematical computations which quickly and accurately predict the strength and weakness of designs and plans. There are bigger benefits to using on-line systems as a communications medium between people.

When one person designs or plans something, he has no communication problem with himself. He can write cryptic notes on pieces of paper and leave them scattered around his desk in a positional notation which only he need understand. He can look in the upper-right corner of his desk for that memo on what he did yesterday. His individual work will proceed at a certain pace.

When more than one person gets in the act, however, a communication problem is created. If a design job is divided up between people, the separate parts have to mesh correctly. If it is possible for the separate actions of individuals to drastically affect the actions of other individuals, then an almost hopelessly confusing tangle is possible. Imagine us designing an aircraft. My responsibility is the electrical wiring; yours is the gas tanks. If either of us changes his mind about the position of our part of the design, the other must be informed as quickly and easily as possible. By merely providing up-to-date design information to each user, an on-line design system could be a big help.

We have only just begun to work with computers as communication media between people. Today, by linking remote stations, we can allow one person to "look over the shoulder" of another through a computer. We have yet to combine the functions of the design system and the inquiry system. The ability of many people in widely separated locations to know exactly what is going on has already proved practical in the airline reservation system. It must be included in our computer-assisted design systems.

4. Programming Systems—The biggest interest in on-line systems, judged at least by the noise people are making today, is in on-line programming systems. In the past two years, we have learned a great deal about how to get on-line service for computer users. We have a spectrum of on-line programming systems, from simple and fast to complex but slower. But we are only just learning how to time share our computers. There is a great deal still to be learned about memory sharing and routine sharing. Such techniques will enable more than one user to share the capacity as well as the time of the computer.

Today's on-line debugging techniques are still rather crude. We can now communicate with a computer program in symbolic assembly language if we used such a simple language in writing the program in the first place. It is possible to insert break-points in the program, to stop it when certain conditions arise, and then to examine what went wrong. We have not yet learned to communicate with similar fluency in any higher-level language. We have not yet built the systems—although we could within the next year—which would let us have the same fluency of on-line communication with programs written in higher-level languages.

At the moment, we are still adapting off-line techniques to our on-line systems. For instance, we still see remnants of the card image in on-line systems. If, in a truly on-line system, there is no need for punched cards, why maintain the card image? Of course, we maintain the card image because it is more economical to adapt our existing card-oriented programming systems to our new on-line techniques than to start afresh. Our progress in getting "on-line" can be measured by our success in abandoning entirely old concepts which do not contribute to an on-line system.

We are only just beginning to explore systems where the computer asks questions of the programmer to resolve ambiguities in what it is told. Imagine a situation five years from now when, given a problem to solve, I approach a machine and say, "I have a problem in numerical analysis to solve." The machine asks me a few questions about my problem and decides what the appropriate programming language is to use. I write a few expressions in the language, making, as usual, a few mistakes. The machine asks, in each case, what I really meant—perhaps giving me an interpretation of what I said in different terms. I recognize my mistakes and correct them immediately.

What languages are appropriate to on-line use of a computer? McCarthy claims that programming a computer in English is like flying an airplane with reins and spurs. But, programming a computer in English is a much more reasonable proposition for me than programming it in Hindustani; just as flying an airplane with my hands and feet is a much more reasonable proposition for me than flying it with my elbows and knees, because using my hands and feet seems more "natural". To improve all our on-line systems, we need more and better languages of communication between the man and the machine which are "natural" in the sense that they are easy to

use and fit the task. Why can't I write mathematical equations which look like mathematical equations and have the machine accept, compile and perform them? Why can't I describe network problems to the computer by means of the picture showing the network? Why can't I, in filter design, place poles and zeros on the complex plane? The answer in each case is: I can in principle, but not in practice. As yet, the techniques which let me do these things are not widely used. The prospect of the next five years is exciting because there is so much that we now know can be done, so much that we even know how to do, so much that we can put into use by just taking the time and trouble to do so. The prospect of the next five years is exciting because we will be finding out which of the things that we know how to do are actually worth using, which are economically feasible, and which are truly useful.

5. On-Line Problem-Solving Systems—The time is ripe to collect the techniques of pattern recognition, process control, and heuristic programming together to gain a new capability. There are simple tasks to be done in places such as space where humans cannot go, or even communicate, which machines based on these three techniques could do automatically. In the near future, we can expect such machines—"automata" if you will—to come into experimental use.

The development of automata will be good for the contributing disciplines. Pattern recognition workers have taken little account of systems which, by acting, can gather additional information to clarify ambiguous patterns. Have you ever had to move your head to complete your inspection of something? Of course you have. Similarly, we must learn how to make computers actively seek information about their environments. In the context of visual pattern recognition, this implies "taking a better look." In the context of man-machine interaction, this implies that the machine might pose a question to the man.

Process control today is little removed from the servomechanism. While it is true that we control very complex processes, the rules used are relatively simple. We think naturally of assembly line balancing, to optimize profit. The processes controlled today are uniform. In fact, industries which deal in non-uniform products have had some difficulty in automating. For instance, an automated coal mining scheme failed because of the variation in the size of the coal seam. Automated shoemaking is made difficult because of the variability of leather. Pattern recognition and heuristic programming can contribute versatility to process control. Opening up this new area for application of heuristics will stimulate our heuristic techniques.

6. Instrumentation—On-lineness is a two-way street. Not only can we put computers on-line with human beings, but also we can put human beings on-line with computers. We can devise and build *instrumentation* to let a human see what is going on inside the computer. The information processing industry is uniquely wanting in good instrumentation; every other industry has meters, gauges, magnifiers—instruments to measure and record the performance of the machines appropriate to that industry. Think of a gasoline engine under test. The test stand bristles with devices to measure temperature, speed, vibration, fuel consumption, and so-on. Civil engineers have even instrumented a huge block of concrete, a dam. Gauges embedded in the concrete measure strain, temperature and humidity deep within the structure. How else could you find out what the internal conditions of the structure are?

Now think of a computer program under test. We run several sample problems, and check the answers. We rarely bother even to measure the time it takes to run the program. Certainly, we do not bother to take statistics on the number of times the various program paths are taken. Yet, in the information processing industry we are uniquely able to make instruments out of the very same stuff, computer programs, out of which the device being tested is made.

Some simple computer program instruments have been made. I have used a program which interprets the program under test and makes a plot of the memory address of the instruction being executed versus time.[1]* Such a plot shows the time the program spends doing its various jobs. In one case, it showed me an error which caused a loss of time in a program which nonetheless gave correct answers. At Stanford University, a program which plots the depth of a problem tree versus time was used to trace the operation of a

*Numbers refer to bibliography at the end of each paper.

Kalah-playing program. Kinslow printed out a picture of which parts of memory were "occupied" as a function of time for his time-sharing system[2]. The result shows clearly the small spaces which develop in memory and must remain unused because no program is short enough to fit into them. Project MAC is using a display to show the dynamic activity of jobs within its scheduling algorithm. Watching this display, one can see jobs moving to higher and lower priority queues as time passes.

Such instrumentation is not in widespread use. We can and will develop instrumentation which will be automatically inserted at compile time. A user easily will be able to get a plot of the various running times of his program. Think of the thousands of dollars saved by tightening up that one most-used program loop. Instrumentation can identify which loop is the most used.

CONCLUSIONS

The future of on-line systems depends a great deal upon the future of off-line systems. There is a lot of talk these days about a semi-automated mathematical laboratory in which a mathematician could prove theorems that he could not prove without computer assistance. How about having the computer prove the theorems all by itself? Suppose the artificial intelligence people make a machine which can, in fact, prove new theorems all by itself. What then becomes of our semi-automated mathematical laboratory? It's useless. Suppose we finally write a computer program which is able to write computer programs. Suppose we could state our problems to a computer able to program itself to solve the problems. What then will become of the on-line programming system? It will be unnecessary.

Today, we are in a very exciting period when interest in on-line systems is very high. Our great surge of interest in on-line systems cannot last forever. What is next? What comes after the on-line systems? Perhaps we shall return to off-line systems as our capability grows to have machines become better able to do things all by themselves. Probably it takes a very large computer to solve useful mathematical theorems automatically. But, it is nonetheless likely that we shall eventually build such a system. In the past, there have been cycles in our interest in on-line systems. In the early days, on-line use of computers was common because no one knew anything else to do. Then there were the bleak years of insulation between users and computers to gain computing "efficiency." Now, we are again in an outburst of interest in on-line computer systems.

In the future, also, there will be changes in the emphasis on on-line systems. In five years, on-line programming systems will be commonplace, and a conference on on-line systems would be out of place. Research interest in on-line systems will have faded, although application of them will still be widespread. Perhaps general-purpose automatic problem-solvers will come into use soon after that. If so, even the use of on-line programming systems may decrease.

Eventually, the process control on-line studies and the automatic problem-solving work will come together to make automata. Computers will then be truly on-line with the physical world in the same sense that we human beings are on-line with the physical world. Once again, there will be a resurgence of interest in on-line systems. What I am predicting is that today's interest in systems in which a man and a machine get together on-line will be replaced in the distant future by interest in systems in which a computer gets directly on-line with the real world, sensing and interacting with it directly through transducers. The "real world" with which such systems interact will include human beings, of course.

CHARGE

We are embarked on the greatest adventure of all time. We believe that human beings are individually valuable and have inalienable rights. We believe that human beings are not to be used as slaves. We must find something else to give us the freedom of action which we call leisure. We turn, of course, to the machine. It will do our work for us so that we may be free to do only things which we wish to do. We will be free to exercise our creative impulses.

REFERENCES

[1] J.C.R. Licklider and Welden E. Clark, "On-Line Man-Computer Communications", *AFIPS, Spring Joint Computer Conference Proceedings, 1962.*

[2] Hollis A. Kinslow, "The Time-Sharing Monitor System" *AFIPS, Fall Joint Computer Conference Proceedings* Vol. 26, Part I, 1964.

Dr. Walter F. Bauer*

On-Line Systems—Their Characteristics and Motivations

INTRODUCTION

THE CURRENT CONSENSUS among computer professionals is that on-line applications represent the wave of the future. The existence of this symposium itself stems from that conviction. However, as with all new subjects, some important basic questions arise. It is well to contemplate what on-line computing is and why it is becoming so important.

First, some estimates and forecasts (**Figure 1**): on-line computing probably represents 1 per cent of the total computer activity in the country today. It will probably represent 50 per cent in five years. Within ten years it will probably represent nearly all computer activity. This symposium and our discussions come, then, just at the beginning of this new "revolution".

Modern computers are about fifteen years of age. The computer profession has undergone much strife during its formative years but now has reached some degree of structure, standardization and predictable growth pattern. Everyone in the computer world knows what subroutines, assemblers and simulation programs are. There is even a rather universal acceptance of the difference between an assembler and a compiler. However, just as this status is being reached, interest has rapidly developed in drastically new approaches to computer use. We are now confronted with new words and techniques: time-sharing, real-time, on-line, and multi-programming. In view of this, it seems appropriate not only to define these terms, but also to discuss the total structure of on-line computing in an attempt to interrelate the various aspects of this new era. This is the major objective of this paper.

Another objective is to examine the motives of those advocates of on-line computer use. Is such use cheaper? What does it gain for the user? What is to be gained by on-line computing versus batch processing?

But the prime question may well be whether on-line computing itself is basically new, or does it represent a natural extension of older techniques? It is interesting and instructive to trace the evolutionary paths which brought us to our present capability (or desire for capability) of on-line computing.

Last, but not least, is the series of interesting questions dealing with techniques and technologies which inspired or were made necessary by on-line computing. The implications to the user, the programmer and the machine designer are profound, but not unattainable. They should be spotlighted early to allow time for balanced development of all their facets.

DEFINITIONS AND STRUCTURE

First, it is the proper time for the computing field to rid itself of old fashioned words and adopt more meaningful terms. The phrase "real-time" is itself a meaningless expression. This hyphenated expression was important a decade ago when a computer was lashed to instrumentation or tied closely to the outside world. The term was used to describe those tasks that needed to be locked or synchronized on a second or millisecond basis to some real time occurrence. As applications of this type branched out, the term became more and more inappropriate. Question: is the SABRE system for airline reservations appropriately

*President, Informatics Inc.

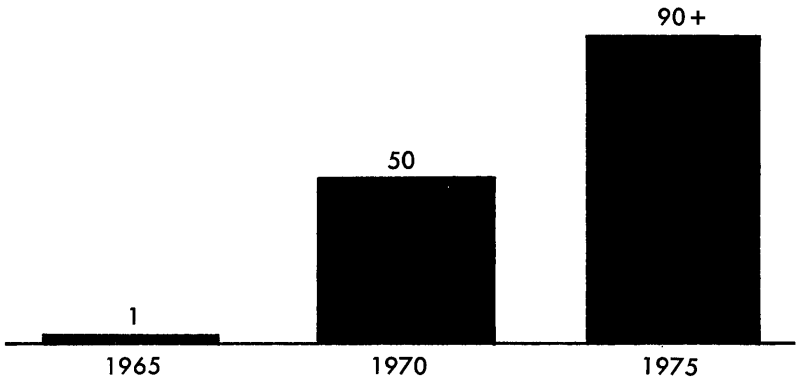


FIGURE 1

ON-LINE COMPUTING GROWTH IN UNITED STATES

called a "real-time" system? The fact that lengthy papers[1]* have been written attempting to define real time is, in itself, ample evidence that this misnomer for an application category and its description should be straightforward and simple.

The word *on-line* has been chosen as the title for this symposium. It is more meaningful than "real-time". It seems that definitions are long lasting and meaningful only if they are simple. With this in your minds, the following definition of *on-line computing* is put forth for your consideration.

> "On-line computing is the efficient use of a computer in a system in which the computer interfaces with man or other machines to which it reacts in receiving and supplying information."

Let us not attempt to define on-line or real-time by some abstract reference to the passage of time or the urgency of the receipt of data, but rather, let us define it in terms of the environment of the computer system itself and the manner in which it is used.

The above definition should withstand your scrutiny.

Analog/digital hybrid systems are on-line computing systems since the analog computer itself is a clever machine which, like man, receives and supplies information.

An airline reservation system is on-line since the computer reacts to signals generated at the ticket office via an input device.

Systems oriented towards scientific problem solving by use of a console are again on-line, since the console itself is the interface which, in turn, receives information from, and gives information to, the human who has a need to know.

The purist may argue that on-line computing then refers to *all* computer systems, since they must all have devices such as card readers and punches to give and receive information. We can avoid this weakness in the definition by insisting that interfaces with men and machines do not include conventional input/output equipment. We can also insist that the words "to which it reacts" rule out conventional input/output since, in those cases, the computer itself controls or drives the input or output process instead of reacting to it. In other words, for on-line systems, the computer is embedded in a system, and the part of the system outside the computer synchronizes the system. The signals to which

*Numbers refer to references at the end of the paper.

the computer reacts are frequently random. This is in contra-distinction to those applications where the computer itself synchronizes input/output equipments.

Referring to **Figure 2**, it seems natural to divide on-line computing into two major areas: man/machine oriented applications, and instrumentation-oriented applications. In the instrumentation-oriented case, the computer is locked into instrumentation to which it reacts; in this case human participation is incidental. On the other hand, in the man/machine oriented case, instrumentation primarily enables man to "talk" to the system. The system is necessarily oriented to the console and to the men who operate it.

We should hasten to add at this point that in creating definitions and meaningful structures, the obvious weakness is that many systems are not pure but, in fact, blended. In reality, larger systems are both instrumentation-oriented and man/machine-oriented. A large scale communication system, for example, will probably have an elaborate man/machine subsystem which allows extensive monitoring of message processing, or for human intervention for pathological cases which might arise.

Instrumentation structured systems might further be broken down into "simulation" and "discrete" types. A simulation type is one which is closely synchronized by, or in concert with, events as they are happening. Now with the discrete type, the system reacts to signals which are less frequent and are mostly random, such as those described by Poisson distributions. The latter are systems in which queues form and service may be relatively unpredictable, and may vary considerably relative to demand.

But we are here to examine the man/machine systems. Therefore, let us look with greater precision to applications and systems where the man is closely interacting and reacting with the system.

Referring again to **Figure 2**, there seem to be three major areas for man/machine applications; these are problem solving, programming and computer use. In problem solving, man wants to have the computer carry out complex processes whose parts are chosen and initiated by him. The computer is solving the problem in the sense of carrying out the detailed manipulations required. The man acts more as a control system monitor; he controls the pieces of computation. One of the best

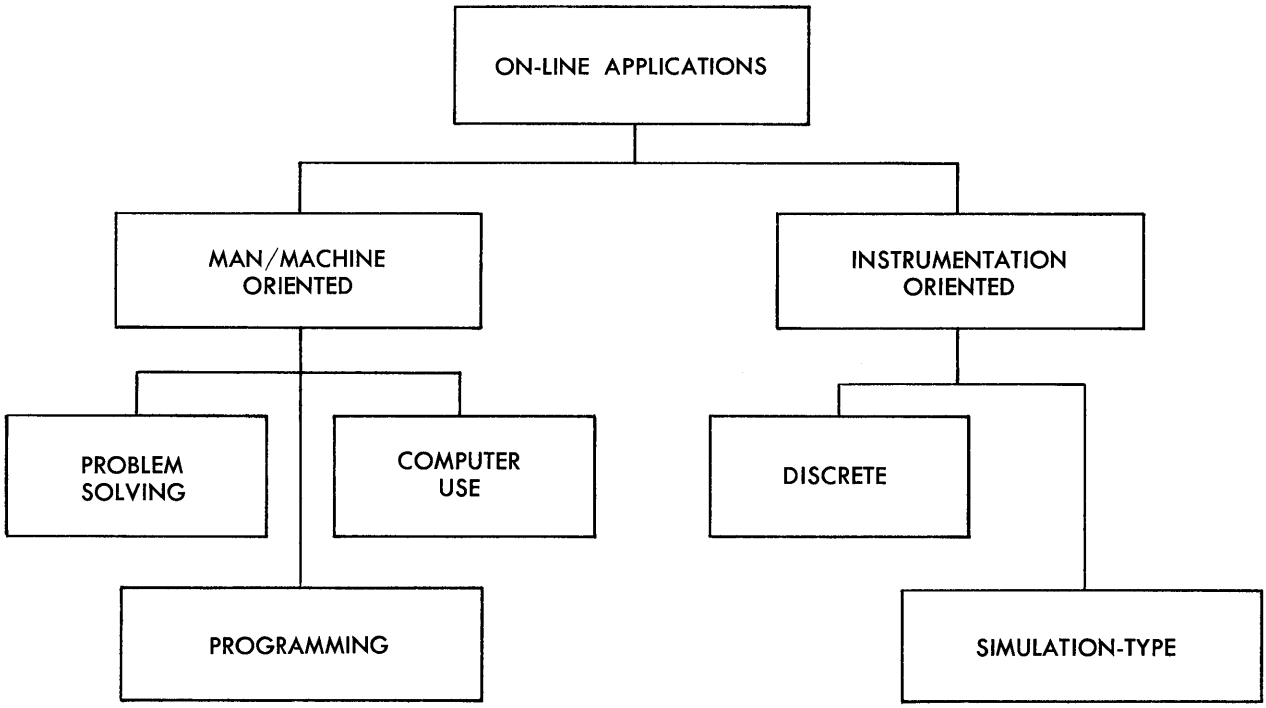


FIGURE 2

TWO MAJOR AREAS OF ON-LINE COMPUTING

examples in problem solving applications is that developed by Culler and Fried[2]. In their application, the computer can perform a wide variety of mathematical operations on a collection of points on an interval.

On-line programming is the second area identified. In this application, the man uses the computer to develop an end product; an object program which accomplishes a prescribed known processing result. The programmer has used the computer to assist him in the process of selecting subroutines, preparing programming pieces for proper embedding into larger systems, for supplying data, for correcting the format of his instructions, or for examining the logic of his program structure. There seems to be little of this being done now as I have defined the problem here. Most on-line programming systems involve simulation of problem-oriented language statements which comes under the heading of "computer use", as described in the following paragraph.

The third area of on-line applications is heavily oriented toward man. This refers simply to the use of the computer by the man. In this application the computer can be considered as his strong right arm. The problem is solved by the man assisted by the machine. In input and output of data, for example, the computer is assisting the man in establishing formats, priorities, data locations, and so on. Another application is the question of a data base where the man is asking questions, and succeeding questions depend on previous answers. Still another application in this category, is the control and monitoring of large scale systems.

While we are in the business of defining and naming processes, there is another term which needs attention. We have already dissected "real-time" and have concluded that it is an old-fashioned phrase which has little meaning in a modern sense. Another unfortunate phrase is "time-sharing". This phrase is usually used to describe the simultaneous use of a machine by a number of programmers or analysts. Many descriptions appear in the literature[3,4,5,6]. However, in these systems it is not the sharing of the machine which is the most important. Rather, it is the orientation of the machine to the human. To be specific, time sharing comes into the picture because this is the only current efficient way of using a computer in close cooperation with a human, since the machine would not be used efficiently during the "head scratching" period of the operator, and in a "time shared" system, many operators can use it simultaneously. Thus idle time is reduced. Time sharing is a result of the fact that there is the man/machine orientation. To illustrate, the most important thing about an automobile is that it supplies transportation. The fact that it is efficient to have round wheels and a gasoline engine is overwhelmed by the transportation factor. And so with computers. Thus, the expression "time sharing" puts the emphasis on a secondary characteristic and is, therefore, not good terminology. I offer that "on-line" is a much more accurate and representative name.

MOTIVATIONS FOR ON-LINE SYSTEMS

In general, the motivation for on-line systems is to make the computer a more powerful tool. The computer is being made into a more powerful tool by building it more responsive to the user—more responsive especially in respect to type of information obtained, and time required to get it.

These systems greatly increase the efficiency of the user by giving him information which reduces red tape and mundane clerical operations. They bring the user closer to the data inside the computer and make it more accessible to him. They give the user only the information he needs, when he needs it. In other words, with on-line systems there are no lengthy outputs from which the operator must laboriously select the desired information. The on-line concept is the skeleton key to the files.

However, it behooves us to look carefully at on-line systems from the standpoint of the areas of assistance which the computer can provide to the user. If this is not done, the danger exists that we would expect too much from on-line systems; in our enthusiasm for all their merits we could create a considerable disenchantment among over-sold users.

In the situation of man interacting and reacting with the cool gray computer, the appropriate question is, "How can the computer help the man?" The following four statements about this assistance should represent a mutually-exclusive and mutually-exhaustive set of assistance areas. They are:

1. Correct an input.
2. Accept a query or the specification of an operation, then perform the required de-

sired operation which should provide the proper information to the user necessary to accomplish the next step.

3. Accept and appropriately handle information to enable the computer to interpret correctly future information which it may receive from the user, or
4. Direct, on a step-by-step basis, a procedure for information input and output.

Let us examine briefly each of these areas.

To correct an input and to immediately signal the man in the loop for a correction or the need for a correction, is an important time saving factor. Many needless hours of "turn around time" can be saved by finding out immediately that a transcription error exists. This is just one example.

One of the important aspects of computer use is in the area of computer aided processes. The man is carrying out a number of logical steps and needs the extra strength of the computer to help him in each of these steps. Because the specification for each step depends upon the results of the previous step, there is need for fast response. Consider, for example, a military commander asking questions about the status of enemy forces. It should be easy for you to imagine the series of interrelated interrogations.

While carrying out the procedures in many on-line systems, it frequently becomes obvious that certain procedures should be changed. For on-line problem solving, for example, if it is found in solving certain problems or in solving certain levels of problems that a procedure, (e.g., multiply by cos X and integrate over range 0 to 1) occurs frequently, it can be specified as a standard instruction, and the computer will react appropriately each time this instruction is received.

Computer-directed procedures are an important aspect of on-line computing and are little understood or appreciated. For example, they allow a complex query to be asked of a machine under the control of another machine and with the assistance of a third machine.

One thing should be borne in mind always about on-line systems. They increase the burden on the computer so that the efficiency of the man can be increased. We must realize always that a penalty or a price is extracted for each increment of increased human efficiency; more computer time, more complex computers, greater input/output equipment, and increased console costs. The tradeoffs undoubtedly favor ever increasing on-line capability. However, no system design should occur without recognition of the prices demanded.

COMPUTER-LEAD PROCEDURES

The importance of this area and its apparent lack of appreciation by computer users suggests more attention should be given to the definition and the benefits which can be derived.

Consider for example, the process of complex interrogation of a large data base. The computer must be an active participant in the process or the operation becomes unwieldy. Consider the functions as shown in the center panel of **Figure 3.** The user must perform the selection, but functions must be performed relating to the logic or syntax of the request and to the consultation of a dictionary or format specifications. These in turn require a look-up operation to files which provide the required data. In manual operation, as shown, the computer performs only the process of retrieval and presentation. Obtaining the procedure to be followed by consulting a comprehensive operator's handbook is left as a burdensome human-only operation.

In the console-computer automated operation the only function left to the operator is that which *must* be left to him; the selection. The computer leads him by the hand, hopefully by the proper digits, down the rocky procedural path.

As a simple example of the principle espoused, imagine that a military commander wishes to have information about POL (petroleum, oil, and lubrication) resources and airfields in a certain section of the country. Also, suppose he wishes to have a list of airfields in five western states which have a POL availability of 80 percent after an enemy attack. It is totally unacceptable and time consuming for him to make the request to a technician who would then transfer the information to an obscure code or punch it on a card. Rather, he makes the request to a staff officer who directly questions the machine.

Consider the following as a sample procedure. The staff officer may specify to the machine that he is interested in "installations" and chooses, from a list of installations which the machine gives him, the category

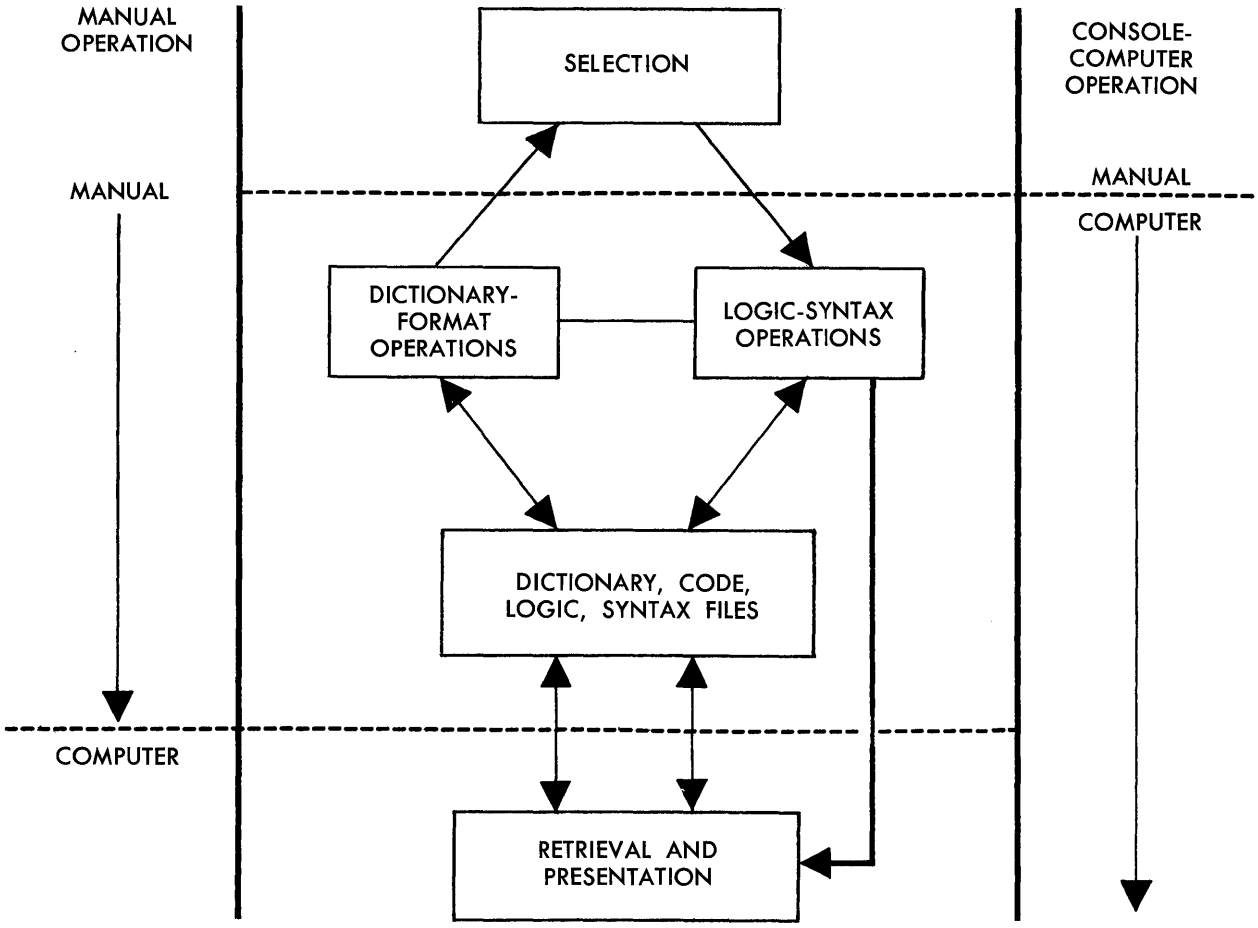


FIGURE 3

COMPUTER INTERROGATION PROCEDURES

"airfields". He then tells the machine he is interested in "resources" and the machine provides him with a list of resources from which to choose. A similar procedure takes place with respect to the geographic location. When he tells the machine that he is interested in an availability of 80 percent, the machine responds with a form to fill out. The officer enters the number 80 on the form and the list is printed out.

The format of the request is natural. It is neither highly stylized, nor codified. The staff officer making the request and pushing the buttons is himself a military man who understands the commander's request and the reasons for it rather than the technical details of how to make the machine accept or respond to the request in its complex electronic way.

The benefits of computer-lead procedures, however, are not limited to interrogations of the data base. The inverse procedure benefits equally man and machine. Consider the input of data which is relatively unstructured. The computer, of course, must finally accept and file the information in a highly structured form so that it may be retrieved efficiently. The computer can direct a procedure which allows the human to input the data in the order and in the form which the machine can accept.

THE EVOLUTION OF ON-LINE SYSTEMS

The advent of on-line computing systems has not blossomed suddenly, nor has it sprung full grown as a technical revolution. Rather, it can be viewed as a normally evolving ca-

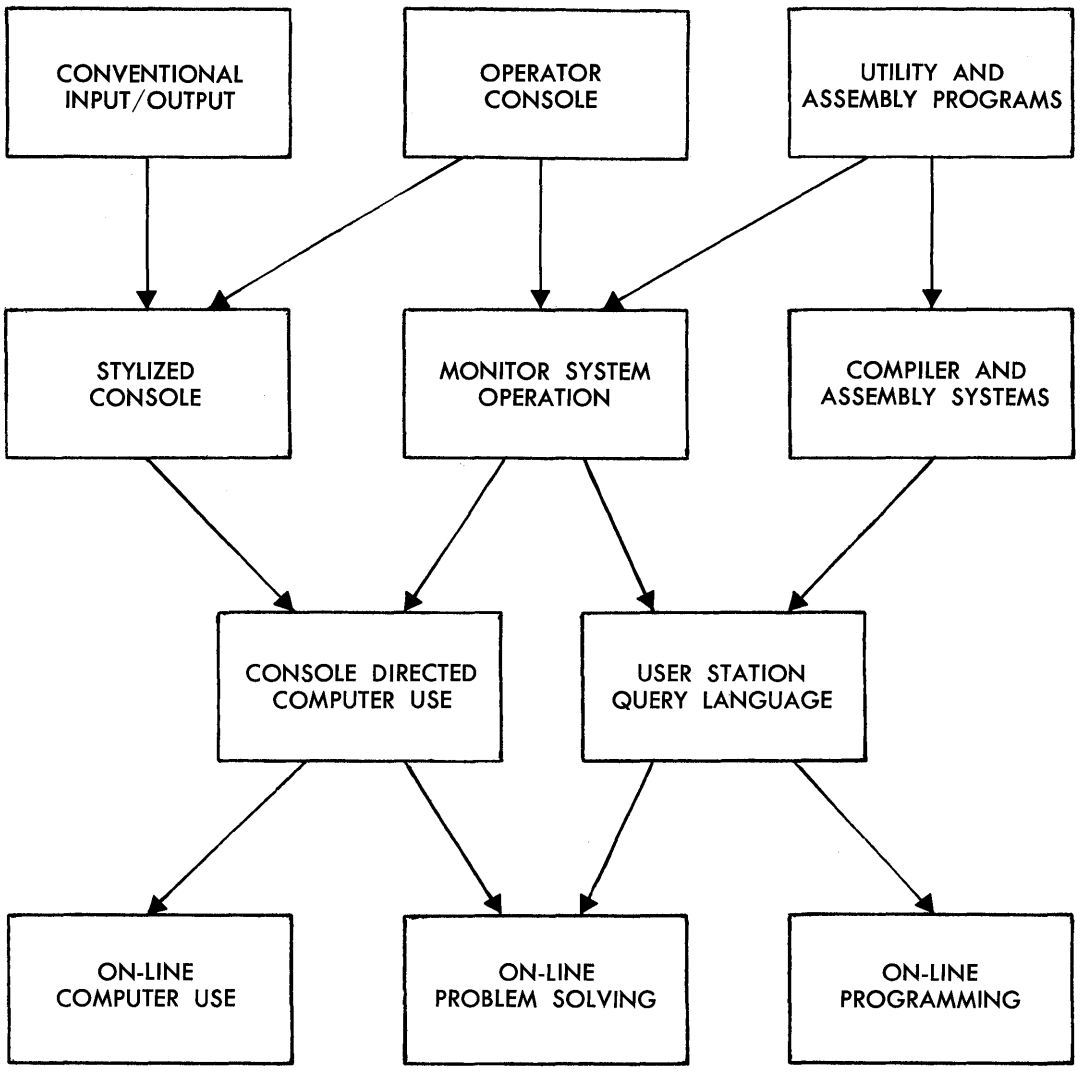


FIGURE 4

EVOLUTION OF ON-LINE SYSTEMS

pability. The capability and potential has been there and recognized for some time; it is the increased *attention* being given the techniques which is sudden and dramatic.

To illustrate the evolutionary process, consider **Figure 4.** Portrayed there is capability, increasing from top to bottom accompanied by passage of time. In computer use in the early '50s it was the conventional input/output, the operator console and the utility and assembly programs. People who used computers and sat at consoles for long hours could not help but conclude that there was a faster, better way to give and receive information from an operating computer. The SAGE system[7] and the early output device using a cathode ray tube were examples. Also, operator console procedures became more sophisticated. Many system designers concluded that if you could give information at the console for diagnosing hardware failures as the maintenance men did, then why could not the user give complex instructions to the machine dealing with the processes and procedures being carried out by the machine? These were illustrated in a number of systems which have been described in literature[8]. At the same time, strides were being made in non on-line

areas with the development of new, advanced compiler and assembly systems in step with powerful computer-user languages.

System designers, naturally, borrowed heavily from the techniques being developed in stylized console and monitor system operation to develop console-directed computer use. In other words, instead of submitting information to the computer via punched cards and waiting for the results to appear on the printed edge, these designers suggested keying in data directly into the computer at a console or user station and receiving back almost immediately the information at either location. The computer actions being directed by the human then became very diverse and flexible. Many of these have been highlighted in the literature[9,10], especially those relating to exotic military systems.

About this time people took still a different –but related–approach to implementation of query languages at user stations[11]. The query language approach borrowed heavily from the technical developments of problem-oriented languages. The resulting query language was highly formatted, bearing many family resemblances to the conventional compiler languages which were designed for immediate question and answer on-line use.

These two developments gave rise to what seem to be the present three streams of effort; on-line computer use, on-line problem solving, and on-line programming.

PROGRAMMING STRUCTURE AND FUNCTIONS

There are five major parts to the programming system of an on-line system: the console and communication programs, the executive, the utility programs, the operating programs, and the data base. These are shown schematically in **Figure 5**, along with the information flow among them.

The console and communication program provide the linkage between the human and the system with a display console as interface. In some cases where information is flowing to and from communication devices, programs to handle these functions would also come under this category. Three major functions are included: input/output data buffering, input/output data formatting, and operator logic controlling. Data which is carried into the system at the console must be temporarily

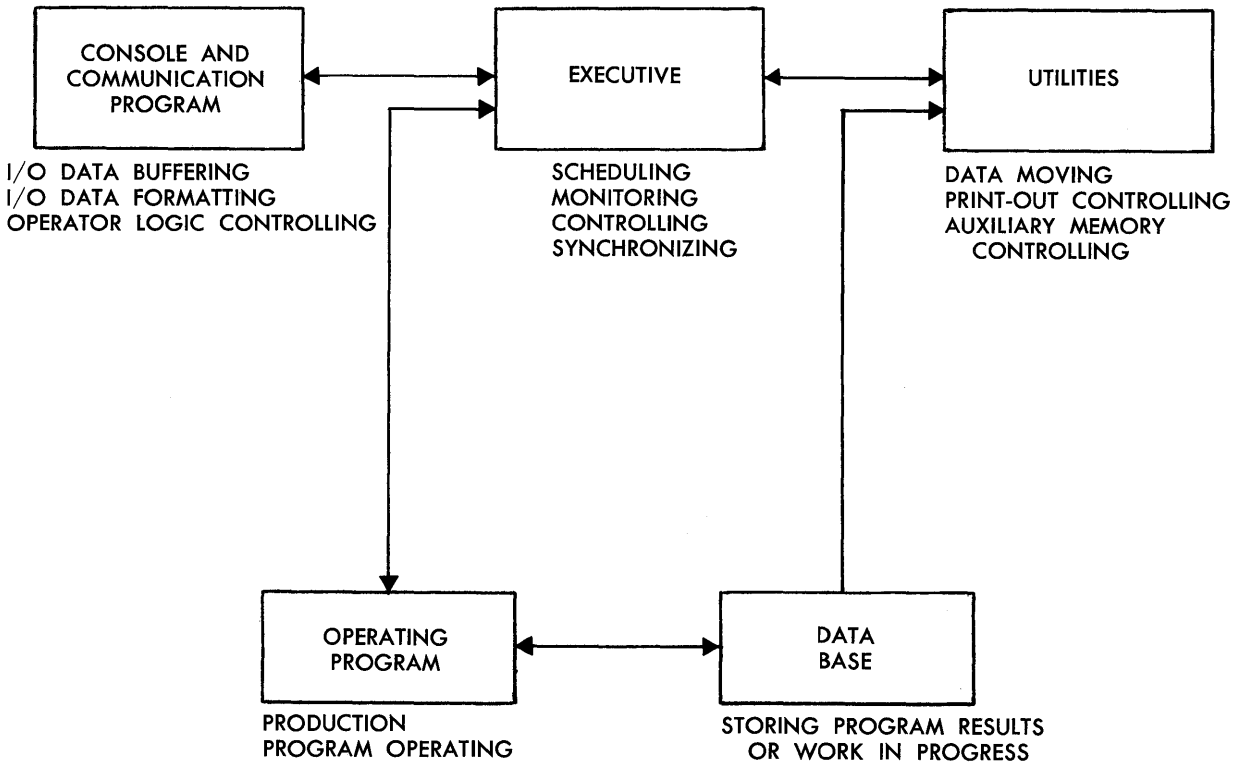


FIGURE 5

MAJOR PARTS TO ON-LINE PROGRAMMING SYSTEM

buffered. Frequently a number of characters are buffered until an "end of message" signal is provided, after which the total message is sent to the executive program. Also, format changes are frequently required to shape information to the form that the executive can accept.

One of the newer concepts in operator consoles is represented by operator logic controlling. Under the topic computer lead procedures described above, there is console procedure logic which is implemented by the console and communication programs. These programs lead the operator; the steps depend on the state of the system and the particular operation being performed. They may lead him by providing information on the next step, by informing him which next console steps are permissible, or by signaling him when a console step has been initiated which is not permissible. Properly designed, the console and communication programs are flexible and modular, and they can be modified easily to accommodate changing procedures.

The executive program provides the basic control for the system. It schedules all work to be performed; whether that work is to be provided by operating programs or by centralized input/output. It also monitors the entire operation and reacts to any system interrupt which signals the occurrence of an undesirable circumstance. For example, the executive may be alerted when a reserved portion of the memory is about to be used up. The executive also provides the functions of controlling the entire operation; that is, it initiates the various programming pieces and provides them with the operational parameters required to define a requested task. Last, but certainly not least, there is the job of synchronizing. In time-sharing, for example, each operator may receive computer time in cycles of, say, 200 millisecond increments, allotted to him by the executive in synchronizing the system operation.

The utility programs provide three basic functions: the movement of data within the system required by time sharing or pooled procedures, the controlling of the printout of information on a pooled basis, and the controlling of accesses to auxiliary memory.

One of the most challenging problems in on-line systems is the correct design of utility programs which can be considered as overhead programs initiated by the executive program. Data must be moved, for example, from working memory when the user or operator leaves his console. Data is constantly shuffled within the system to accommodate the various modes of system operation and the requirements placed on the system. Similarly, utility programs are necessary for controlling the printout of information when the printout facilities are to be shared by all of the users, as they usually are in an on-line system. Controlling the auxiliary memory is another utility type task since the bulk storage devices are usually shared by all.

The last two parts of the programming system are the operating programs and the data base, the latter referring to the storage of intermediate or final data. The data base communicates primarily with the operating program; for flexibility, there is a secondary link to utilities. The utility routines may, at the request of the operating program working through the executive program, request utility programs to handle certain data in the data base.

Some of the general principles to be considered here are:

1. Utility programs exist for the benefit of each system user and for the system itself. They are not considered part of the operating programs, but rather they are of a general utility nature which is called upon by the executive.
2. The executive is in reality the control device of the system. Nothing occurs automatically within the system without its being initiated and monitored by the executive program.
3. Although the data base is a passive program it is included in this structure because the data base can itself be a computer program which is being operated upon by an operating program in an on-line programming configuration.

THE NEW TECHNOLOGY

On-line systems are giving rise to new hardware and software technology. Computers must necessarily be designed quite differently if they are to work efficiently in these kinds of systems. Also, there are a number of badly needed software and programming techniques to provide efficient and economical system implementation.

Some of the hardware aspects which need attention are as follows:

1. Memory protect hardware. This is a device which allows the currently operating program to be "locked out" of all but one part of the memory. The capability must be dynamic and under flexible control of the executive program, since the allowed operating areas of the memory change on a millisecond basis.
2. Inexpensive consoles. The computer industry must develop inexpensive consoles. A reasonable goal is that the console should sell for under $15,000, that it should have a cathode ray tube or something equivalent, that it should allow for some buffering of information, and that it should have a flexible and adaptable keyboard structure and status information display.
3. Multi-computers. Computers must be designed which allow the incremental addition of modular components, the use by many processors of high speed random access memory, and the use by many processors of peripheral and input/output equipment. This implies that high speed switching devices not now incorporated in conventional computers be developed and integrated with systems.
4. Improved input/output. The entire range of input/output parameters needs overhaul. For example, the random access memory device must be accessible through two to five channels. Channel logic and interrupt procedures must provide greater capability than they do on most present computers.

Of equal significance are some of the new software techniques which need attention:

1. Automatic segmentation. Since running programs will have access to only a portion of the memory, frequent "page turning" will be necessary as the program goes through its major operating pieces. Segmenting of the program can be very difficult if it must be done manually by the programmer. Assemblers or compilers with automatic segmentation or semi-automatic segmentation are needed.
2. Relocatable programs and automatic, dynamic relocation. There is the need to be able to produce relocatable programs, and to relocate these programs automatically and dynamically. A part of a program in auxiliary storage is seldom placed into the same spot in high speed memory from which it came, and the program segment is called into high speed memory under a wide variety of circumstances and conditions.
3. Computer use and programming modification languages. Entirely new languages are needed to allow flexible and powerful use of the computer from remote stations. A standardized set of macros is needed to provide the user with many of the functions he must perform very frequently. Two of the simplest examples are "begin" and "end" instructions. These signal the system to make ready for the programmer's future activities which he may call upon the system to perform, and they signal the end of his activity to enable the system to begin to accommodate other activties.
4. Console utility programs. Much of the new procedure revolves around the display console. Programs and techniques are necessary to allow operator efficiency, and to allow the easy modifications of programs.
5. Scheduling algorithms. Increased attention needs to be placed on the problem of techniques for scheduling the many users with their different priorities. Priorities, for example, can be assigned externally or they can be assigned on a dynamic basis depending on how long the program has been in the system, or how much time remains before the deadline for results.

SUMMARY AND CONCLUSIONS

Virtually all computer applications will become on-line in the next ten years.

"On-line" is much better terminology than either real-time or time shared.

On-line applications can be considered man/machine oriented or instrumentation oriented, the latter breaking down into three major on-line application areas; problem solving, programming and computer use.

It is important to understand how the computer can assist the man in on-line applications, and it is likewise important to realize that a price is paid in terms of computer time and programming complexity in gaining this user efficiency.

On-line computer developments are in reality normal evolutionary steps which developed from early console and monitor system opera-

tion and the initial SAGE-type display consoles.

There exists a canonical structure for the programming system of on-line systems which consists of five major pieces; console and communication programs, executives, utilities, operating programs and data base.

There are a number of challenging aspects in computer technology generated by on-line systems. Challenges in hardware designs must result in hardware efficiencies to meet the new software techniques and then the challenge of efficiently blending the "hard and "soft" for a truly efficient system.

Respectful attention by all computer users to these essential factors or opinions is necessary for the new technology to keep pace with the demands and the potentials. Those computer professionals who are keenly aware of the problems and needs, and who continue offering improvements, will find the challenges stimulating and their efforts well rewarded.

REFERENCES

[1] T.B. Steel, "The Fabulous World of Real Timeland", *Datamation*, March 1964.

[2] G. T. Culler, and B. D. Fried, "On-Line Computations and Faster Information Processing", *IEEE Pacific Computer Conference*, March 1963.

[3] J. I. Schwartz, E. G. Coffman, C. Weissman, "A General-Purpose Time-Sharing System", *Proceedings of the SJCC*, 1964.

[4] A. J. Critchlow, "Generalized Multiprocessing and Multiprogramming Systems", *Proceedings of the Fall Joint Computer Conference*, 1963.

[5] F. H. Corbato, et al, "An Experimental Time Sharing System", *Proceedings of the WJCC*, May 1962.

[6] J. C. R. Licklider, and W. E. Clark, "On-Line Man-Computer Communication", *Proceedings of the SJCC*, 1962.

[7] R. R. Everett, C. A. Zraket, H. D. Bennington, "SAGE, A Data Processing System for Air Defense", *Proceedings of the EJCC*, 1957.

[8] W. F. Bauer, "Integrated Computation System for ERA-1103", *ACM Journal*, Vol. 3, pp. 181-185, 1956.

[9] W. F. Bauer, "Military Command: A Challenge for Information Processing", *Computers and Automation*, April 1963.

[10] W. F. Bauer, and W. L. Frank, "DODDAC–An Integrated System for Data Processing, Interrogation, and Display", *Proceedings of the EJCC*, December 1961.

[11] T. M. Dunn, J. H. Morrissey, "Remote Computing–An Experimental System", Part 1, External Specifications, *Proceedings of the SJCC*, 1964.

Dr. C. B. Tompkins*

Mathematical Techniques for On-Line Systems†

INTRODUCTION

MY PERSONAL INTEREST might be more nearly the study of mathematics and mathematical techniques *from* on-line systems than the title assigned me. However, the techniques which are currently usable for on-line systems are tools with which we shall expect to contribute to the development of these new mathematical ideas and techniques.

I do not agree that "on-line" computation is almost synonymous with "multiple-console" concurrent computation. I shall consider problems which may impose small computing loads and small communications requirements and are, therefore, also suitable for multiple-user operation; but that will not be my aim. One of the most famous multiple-user devices is the chalkboard (as it might be used in an old fashioned way to teach arithmetic or algebra when a sizable fraction of a class is sent to the chalkboard to work). A pad of paper and a library can provide another multiple-user system, at least under communistic policies of using the paper, and under reasonable rules for circulation of the library volumes.

Use of a single computer by several users concurrently is now one of the exciting sectors in which computers and computation are developing. To the extent that these applications may make an extremely powerful, extremely fast, and extremely economical analogue of a desk calculator, the techniques are roughly those of a desk calculator. The functioning is on-line, however, and such functioning is of interest here.

An on-line computation has one principal advantage over off-line computation—the advantage of being subject to exploitation through implicit decisions by the user. This is independent of the multiplicity of simultaneous uses. The classical view of automatic computation (prevalent in the "remote" past of, say, 1958) included a high evaluation of austerity. Thus, then (and still in some laboratories) efficient and effective use of the available equipment required that sizable bites of the calculation be described explicitly. This was so that every decision in the course of the calculation of each bite could be carried out by the automatic machine in accordance with an explicit rule furnished to it in the coding of the problem. If a process of sequential decision was contemplated, with decisions based on the calculations which had preceded the decision, and if the proprietor of the problem being processed planned to intervene personally at the time of decision, then either his intervention was seriously limited, or the problem would be ejected from the machine to await the decision (and to await the next period of machine operation at which it could be introduced, often a wait of a sizable fraction of a day).

Men pursuing their various goals normally find it necessary to assign various portions of their problems to subordinates in an organization. The assignments normally are of two types: those which can be carried out mechanically without exercise of ingenuity (al-

†The preparation of this paper was sponsored in part by the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government.

*Director, Computing Facility and Professor of Mathematics, UCLA

though a high degree of professional competency may be required), and those which call for judgment and authority and which carry more responsibility than that of blind but competent obedience. The first of these types of assignment might be compared to use of compilers and computers in the austere classical manner; the second is more like on-line computing. The fact that formulas cannot suffice for all the decisions made in the simplest of the pursuits of our lives, might indicate that the inventive process is required in our attempts to solve our (formally) most difficult problems.

Thus, I liken the console operator of an on-line system to a leader of immediately and competently responsive subjects in attempts to solve problems which require exercise of judgement by the leader from moment to moment, rather than from day to day. These subjects comprise the computer system involved in the on-line operation.

In all this, we must recognize that the more we can automate a desired operation, the more time we have available for obscure inventive processes. However, the formulation of a problem in terms of available procedures for automatic processing may impose uneconomical difficulties. The inventor may be able to describe his process of invention explicitly, but it seems likely that the energy and attention expended in this descriptive process will seriously interfere with, or even annihilate, his powers of invention.

The question of on-line mathematics is, then, the question of determining when implicit observation, weighing, and exercise of judgement involving one or more highly competent scientists should be preferred to automatic processing of data, with all observation, weighing, and selection of sequence of operations left to automation operating under explicit rules.

THE MATHEMATICS OF GENERAL SHAPES AND FORMS

In seeking an answer to the question of on-line mathematics formulated above, we are naturally drawn to consider those qualities of concrete and abstract devices which are still efficiently recognized by and reacted to by humans. This immediately suggests "eyeballing," and this, in turn, brings to mind topological aspects of mathematics, again in the classical sense (which this time starts with Gauss and continues through many of the contributions of the less abstract current contributions to topology).

However, there are sensory devices which determine general forms and shapes which can be applied in non-topological ways. It seems to be well established that the eye can detect a straight line segment in a photograph while limited resolution and other imperfections effectively hide the segment from automatic recognition by instruments. The power of the human to hear through noise is equally remarkable. Patterns are recognized by humans of low intelligence, while humans of the highest intelligence find themselves unable to explain the process satisfactorily. Finally, the artistic fitting of theory to fact with a feeling that "a slight modification right there should do the trick" is sometimes known as genius; here it is essential that the theorist be provided with data which makes him believe that the slight modification (or some modification) is needed, and that he be able to obtain experience which lets him "feel" rather than know unequivocally the proper direction and size of the modification.

To list all the interfaces which allow implicit judgement and implicit recognition to enter into mathematical calculations would be undesirable here—the list would be too lengthy and would force me to depart from my subject, and, besides, neither I nor anyone else could possibly present a complete list. We can only try to find the necessary interfaces and mathematics in an implicit way by on-line observation of candidates.

Therefore, I shall turn to some examples of profitable on-line calculations.

THE ISOGRAPH AND ITS TOPOLOGICAL PRINCIPLES

I shall not develop the topological tools needed for calculation here, but I shall describe them as I go along. An elementary exposition of many of them can be found in my paper[1] and the references in its bibliography.

One fundamental problem which has always been present in algebra is the problem of finding the roots of a polynomial. An ingenious on-line device for using topological tools, properly supported electrically and electronically, has been described by R. L. Dietzold[2]. This was an analogue device.

First, I shall describe a proof of the fundamental theorem of algebra because the principles of this proof are pertinent here. In this proof, I shall denote the polynomial by P and its argument either by a number, a symbol z, or another symbol to be introduced later when the polynomial will be evaluated at all points on a circle. I shall assume that the highest power of the argument appearing in the polynomial is n, and that the coefficient of this highest power differs from zero. Such a polynomial is called a polynomial of n-th degree. The fundamental theorem of algebra is:

THEOREM. *If* P *is a polynomial of* n*-th degree, where* $n \geq 1$, *in a single complex variable, there is some argument value for which the value of the polynomial is* 0.

A proof is along the following lines. First, examine the coefficient of the zero-th power of the argument in P. If this coefficient is 0, then the proposition is correct, for then $P(0)=0$. If not, call this coefficient a_0. A polynomial is a continuous function of its complex argument to the complex plane, and hence for any argument value sufficiently close to 0, that is, for any z with $|z|$ small enough, the value of the polynomial will lie as close to a_0 (in the complex plane) as any tolerance specified in advance. At first we specify this tolerance to be $|a_0|/2$. In particular, we take r_0 to be a positive number such that

$$|P(z)-a_0|<\left|\frac{a_0}{2}\right| \text{ whenever } |z|\leq r_0.$$

We now consider a circle in the complex plane containing all values of z with absolute value r, where r is any positive number to be specified. We denote by PR(r) the set of values taken by the polynomial as z takes on all values on the circle $|z|=r$. For $r=r_0$ a curve $PR(r_0)$, restrained to lie close to a_0, is illustrated in **Figure 1**. The important point is that the locus $PR(r_0)$ does not surround the point $z=0$. If we think of the circle $|z|=r_0$ being traversed by a moving point, and of a small *being* at $z=0$ watching the image P(z) as z traverses this circle, then this *being* might have to move his head back and forth like a spectator at a tennis match. But when the transit of the circle has been finished the *being* at the center will again look at the point on $PR(r_0)$ where his vigil started, and his neck will be untwisted.

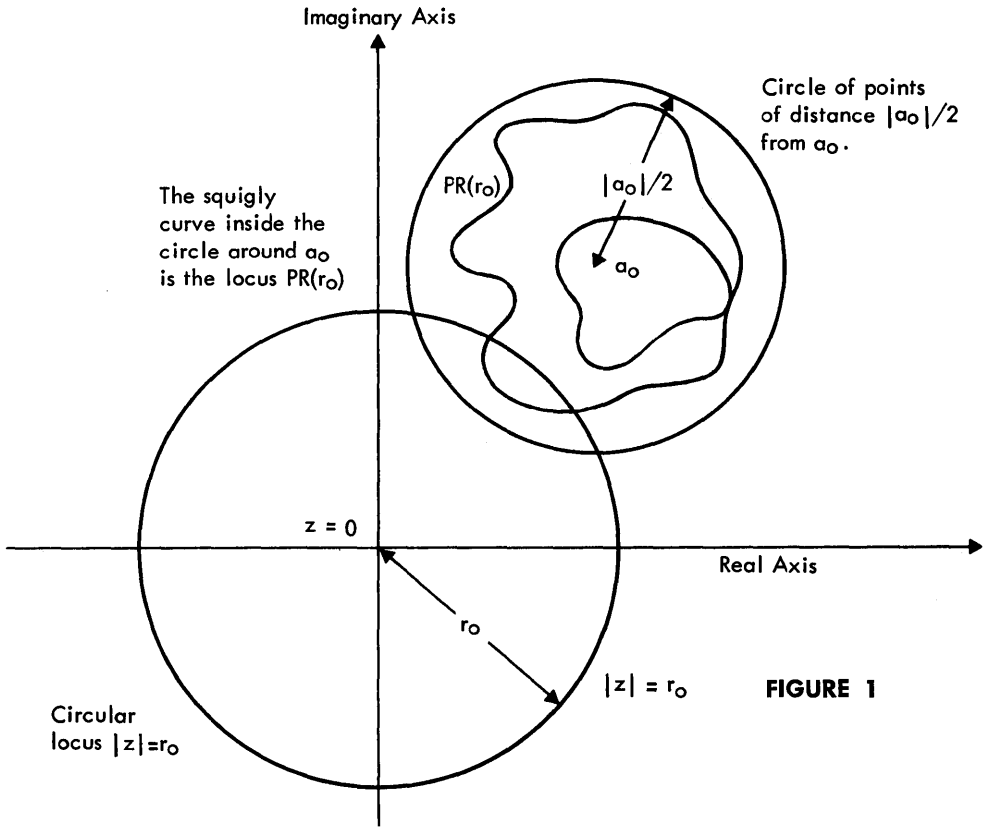
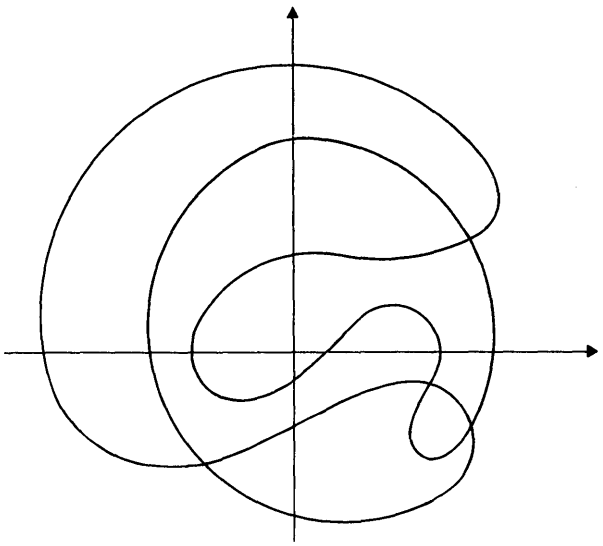


FIGURE 1



A curve looping Z = 0 once.

FIGURE 2

Now consider a value r_1 which is huge, its value to depend on the values of the coefficients in the polynomial being considered. This value is to be taken so that the value of the polynomial is determined to within a few percent by the value of its leading term. Since z^n can be made to overshadow the value of all lower powers of z by any amount, simply by making $|z|$ large enough, this value of r_1 can be attained. If we consider $PR(r_1)$, and view it from the point $z=0$, the locus will surround the observation point, for the locus z^n for $|z|=r$ is simply a circle, of radius r^n and center at $z=0$, traversed n times. Thus, the *being* can now be assigned to follow the generation of $PR(r_1)$ as $|z|=r_1$ is traversed, and some of the tennis viewing exercise will follow, but if the *being's* body is strapped firmly to prevent rotation, the neck will have to twist n times.

The proof of the fundamental theorem of algebra depends on noting that, unless the locus PR(r) passes through the point $z=0$ for some value of r between r_0 and r_1, the number of twists in the neck generated by watching the transit of $PR(r_1)$ must be the same as the number generated by watching $PR(r_0)$. Since these two numbers differ, at least one circle $|z|=r$, for which PR(r) passes through the point 0, exists; and this proves the theorem.

The isograph used this principle in a constructive way. Without going into details, I assert that it is a comparatively easy job to present PR(r) on the screen of a cathode ray tube for any fixed value of r in the interval $(0,1]$ (that is, for $0<r\leq 1$). The operator starts with a low value of r and looks at the screen; if the curve PR(r) does not loop the center of the screen, his starting value is well chosen. He then increases the value of r until one of two things happens; either PR(r) loops the center, or the operator runs against the stop on the knob with which he adjusts the value of r.

If the operator gets to a looping curve, he adjusts the value of r with care, to make sure that the curve actually passes through the point 0 (always, of course, within the tolerance set by deflection irregularities, spot size, and so on). Then the curve $|z|=r$ is traversed slowly (and this again is an easy electrical assignment), and the exact point on this circle which gives the 0 image is found. This is a root.

(In seeking the root, the operator utilizes the three-dimensional character of our space and stands far enough in front of the cathode ray tube to prevent twisting his neck irreparably.)

All roots with absolute value not exceeding 1 are found in this manner. Then the polynomial may be replaced by another in which z is replaced by $1/\overline{z}$ (where $\overline{z}$ is the complex conjugate of z); the resulting expression is replaced by its complex conjugate, and this expression is multiplied by z^n to form a polynomial. This polynomial has coefficients which are the complex conjugates of those of the old polynomial—occurring in reverse order. If the old polynomial values were $\Sigma a_k z^k$, the new ones will be $\Sigma \overline{a}_{n-k} z^k$.

One can argue that the topological looping principle is not necessary for the explanation of the isograph, all that is necessary is that the operator be sufficiently alert to perceive any curve PR(r) which passes through the 0 point. This argument is correct so long as it is convenient to vary r continuously, but in many cases, particularly with current designs of digital machines to be used on-line or off-line, this continuous variation is not convenient, and the preferred method would be to try $r=1$ first; if the result indicates one or more roots z with $|z|\leq 1$, the value $r=\frac{1}{2}$ would be tried, and so on.

I shall insert here some mathematics which might plausibly have developed from on-line

computation; it is at least based on the idea of considering PR(1). This ingenious scheme is due to D. H. Lehmer[3]. It is an algorithm which determines whether a polynomial P has any roots inside the circle with center 0 and radius 1. It is a completely automatic scheme, and it has been coded for convenient operation on a digital computer without operator intervention.

To start with, it should be noted that simple transformations will permit the search for roots of a given polynomial in any circular disc, to be carried out by searching for roots of a related polynomial of the same degree in the unit circular disc centered at 0. All that is necessary is to replace z in the polynomial by (z-c), where c is the center of the circle of interest, and then to replace z in this new polynomial by z/r, where r is the radius of the circle of interest. Lehmer's search converges on roots by pinning them into always smaller circular discs; the general method is illustrated in **Figure 3.**

We assume that Lehmer's algorithm to determine whether any given circle has any roots of P on its interior has been established. We then seek a circle (starting with the unit circle with center at 0) with at least one root interior to it. We then try a circle with the same center and half the radius. Thus, given any circle with at least one root on its interior,

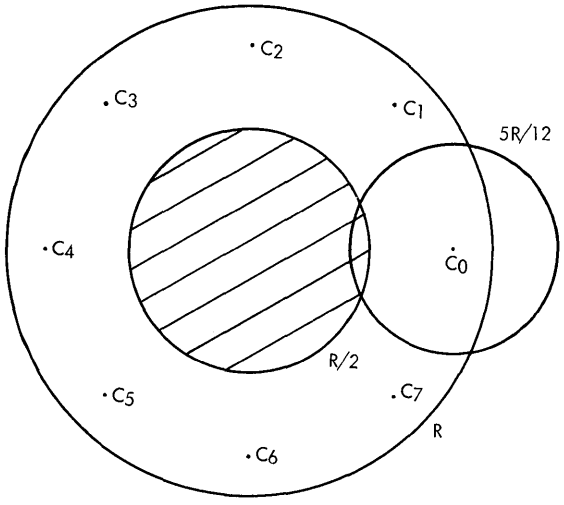


FIGURE 3

we test the circle with half the radius and the same center. Assuming that the center is not a root, eventually we shall arrive at a circle of radius R with roots on its interior, while the concentric circle of radius R/2 has no roots on its interior. These two circles are depicted in **Figure 3.** There are no roots in the shaded region, but there are roots in the annular ring. Lehmer then chooses eight centers, c_0 through c_7 in the figure. He chooses an initial value of 5R/12 for the radius, and he repeats the process for circles centered at each c_i. It is easy to show that these eight initial circles completely cover the annular ring.

Lehmer's algorithm is the following. Let P be a polynomial with coefficient a_k for the k-th power of the argument. Let Q be the related polynomial we constructed above; Q has coefficient $\bar{a}_{n-k}$ for the k-th power of its argument. Now create a new polynomial

$$P_1 = \bar{a}_0 P - a_n Q.$$

This is a polynomial of lower degree than P. If its constant term is negative, then P has a root inside the unit circle with center at 0; if its constant term is positive, then it has the same number of roots within the unit circle as P. In the former case, the tracking down process of **Figure 3** is used; in the latter case the polynomial Q_1 related to P_1 is created, and the elimination of the highest power of the argument is repeated. Eventually, the process leads to a negative constant term, meaning that roots do exist within the unit circle, or to a constant polynomial with positive value meaning that no root exists inside the unit circle.

The simplest proof of this theorem seems to rely on the proposition that the roots of a polynomial are continuous functions of the coefficients—a well-known proposition with few proofs in the literature. Although most published proofs involve study of symmetric functions, the reader can furnish his own proof by index arguments such as those in reference[1]. The proof is trivial if there are no multiple roots. It can be shown that if the constant term of P_1 is positive, then the polynomial $\sigma\bar{a}_0 P - a_n Q$ can not have roots with absolute value 1 for any σ in the interval [0,1]. This argument is mildly topological. A proof of uniform continuity is available in reference[4] pp 156-158.

Thus, the isograph attack, perhaps subtly,

has grown into an attack suitable for either on-line or off-line computation.

However, for functions more complicated than polynomials, it is hard to find a suitable algorithm to place roots inside a simple closed curve bounding a region. Still the turning number method will frequently work. It was applied successfully to solve equations of the type $P(z)\,e^{-z}+Q(z)=0$ by Arnold O. Allen. While a deep study of equations of a particular type might yield a better method of solution, the old isograph principle is certainly the easiest to apply if the worker stumbles on an equation of unpredicted type for the first time. There was no convenient method of generating the graph of $P(z)e^{-z}+Q(z)$ for all z with $|z|=r$ in the days when the isograph was designed, but modern digital computers can construct graphs of functions of a wide class of considerable complexity. One method developed and used with spectacular success has been developed by G. J. Culler and B. D. Fried[5]; this method is described by D. A. Pope during this symposium.

A BRIEF GLANCE AT OTHER TOPOLOGICAL PROBLEMS

The idea of looping closed curves in 3-space goes back to Gauss, and generalizations of the turning number argument above have been developed. Two curves in 3-space are looped if every orientable surface bounded by one is cut by the other. The degree to which two curves are looped is determined immediately by eye, and with difficulty by a computer. The computer would use Gauss's integral formula:

$$J=\frac{1}{\omega}\oint\oint\frac{\begin{vmatrix} x-y \\ dx \\ dy \end{vmatrix}}{[(x-y)^2]^{3/2}}$$

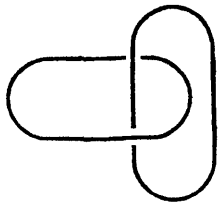
If $J\neq 0$ the curves are looped. In the formula, x and y are parametrized closed curves which do not intersect each other; both x and y are to be considered to be vectors in 3-space. Vector subtraction and scalar multiplication of vectors are intended at appropriate places in the formula.

Figure 4 depicts a pair of looping curves; J computed in accordance with Gauss's formula would take a value −1 or 1, depending on the relative orientation of the parametrized curves. Also in **Figure 4** is a pair of curves which are tangled but not looped; they would give a value 0 in Gauss's formula. Also related to the subject of loops and tangles is the subject of knots. A knotted curve is even hard for a novice to define; it may be defined as a polygon which is not the boundary of a non-intersecting continuous image of a closed circular disc.

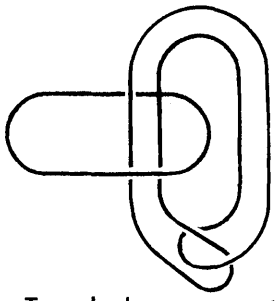
All this might leave a physicist cold. However, it is true that Gauss developed his formula in his study of magnetic forces. It is also true that physicists now deal with concepts that occur in quanta, that some of these physical entities should be determinable by integration around an enclosing surface or hypersurface, and that integrals of the general type of Gauss's form the most general type of integer-valued integrals known. This type is described roughly as the integral of the solid angle in $(m+n+1)$-space swept out by a non-zero vector which is a continuous function of an orientable $(m+n)$-dimensional closed compact manifold. It requires little imagination to believe that one would rather look at the looped curves of **Figure 4** than compute the value of J in the formula.

And, finally, a word about knots. It seems impossible that we can not tell whether two knots are equivalent under reasonable tranformation laws, but derivation of a complete set of invariants seems to be terribly difficult. A pessimistic approach to this problem might have been an attempt to create a knot whose reduction to some "most elementary" form would involve some temporary increase of complexity en route. I shall not stress these matters, however, except to note that modern on-line methods are easier to use than string and that they can be made to record all intermediate stages so that any success in experiments is automatically documented.

I close this part of my discussion with an obvious remark concerning tameness requirements of current systems. Generally on-line displays represent only tame curves. The UCLA version of the Culler-Fried system interpolates linearly between 125 computed points, thus creating a polygon or an arc of broken line segments. Wild curves not adequately described by polygonal approximations currently are infrequently treated by computers. I furnish one example presented by Ralph H. Fox[6]. It is an infinite sequence of crochet stitches. In the sketch I



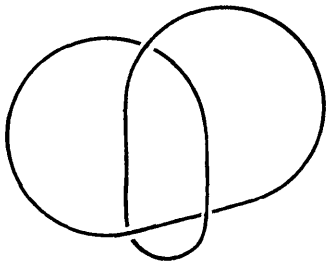
Two looping curves



Tangled curves not looping

$$J=\frac{1}{\omega}\oint\oint\frac{\begin{vmatrix} x-y \\ dx \\ dy \end{vmatrix}}{[(x-y)^2]^{3/2}}$$

Gauss's Formula



A knotted curve

FIGURE 4

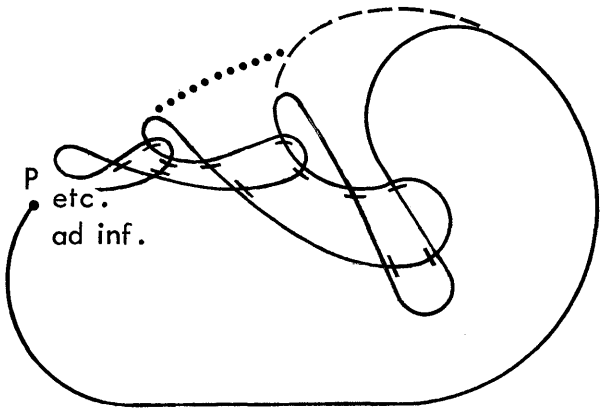
have added a parallel line to the right of the oriented curve when one string passes over another. I have also added dotted lines to show how any finite number of stitches could be unravelled. It seems probable that some subtle method of implicit description would be required before a computer could be called on to aid in the study of such a wild curve.

LOW REDUNDANCY DESCRIPTORS

One of the really powerful tools of mathematical analysis is the construction of convenient complete bases in Hilbert space. The most familiar example, perhaps, is Fourier series. Fourier series uses positive, zero, and negative integral powers of the imaginary exponential function as a complete basis for the space of functions which are of class L^2 (square integrable in the sense of Lebesgue) on the interval $(0,2\pi)$. Fourier integrals furnish the same service for functions defined over an infinite domain. This type of consideration of a function as a vector in a vector space of infinitely many dimensions has great value in abstract studies in that (among other things) it permits the mathematician to prove the existence of solutions to various problems and to describe these solutions, at least in principle.

A great practical value is also realized, for many functions described by Fourier series are described sufficiently well for practical purposes by their first few Fourier coefficients.

An example of this practical value is the familiar study of bandwidth in connection with radio transmission on a frequency in a crowded spectrum. Here, the harmonics can be attenuated by properly chosen filters, and the whole study leads (through a development which is familiar to or easily available to all readers of this paper) to fairly efficient



The wild knot of R. H. Fox

FIGURE 5

use of the available frequency spectrum in single side-band suppressed carrier transmission, magnetic tape recording of digital information, and so on (but not yet for the internal structure of digital computers!). The point is that the Fourier description of a function presents partial descriptors of the function which are adequate for the purposes at hand, and that only a small, wisely chosen part of this Fourier description needs to be computed or used. This part furnishes a description of high efficiency (or low redundancy) for the purpose intended.

Various practical considerations also enter into this picture. First, we should note that we have no access to really existing physical entities extending in time or in space both to $-\infty$ and to $+\infty$. Secondly, we have no perfectly periodic functions. In response to these philosophical objections we reply, of course, that our instruments are far from perfect, and that their reactions to the existing stimuli are indistinguishable from what their reactions would have been to the idealized stimuli which cannot exist. Thus, we stretch or contract quasi-periods of quasi-periodic functions where the actual application is insensitive to minor local variations in rate.

Still considering Fourier analysis, we turn to the Gibbs phenomenon. The Fourier series might converge in the mean (that is, roughly in the sense of power or energy) but still miss the function it should approximate rather badly at some points. In particular, near a discontinuity with different left or right limits, the partial sums of the Fourier series overshoot by a sizable fraction of the jump at the discontinuity, this overjump is bounded away from 0. This phenomenon was noticed by Gibbs and probably by workers before Gibbs, and it undoubtedly has been rediscovered many times after the time of Gibbs. Untold labor would have been saved if the Culler-Fried on-line system had been available. Dr. Culler once demonstrated the phenomenon to me in about one minute starting from scratch, without having considered the problem before so far as its connection with their on-line system is concerned.

Another application in which small variations in rate of generating a pattern can sometimes be ignored is in the study of electrocardiograms. Briefly, the electric potential difference measured by a pair of electrodes attached to the skin shows one large sharp spike and several lower wiggles during each pulse cycle of a normal person. If the patterns of the pulse cycles are similar except for duration of the cycle, one might try to describe the normal cycle by carrying out a Fourier description of one chosen cycle. If this is done, it turns out (not surprisingly) that a discouragingly large number of components is required to get agreement. This is a phenomenon related to the large amount of spectrum usurped by any suddenly changing signal. (Radio transmitters are required to soften their key clicks by filtering out the higher harmonics, and a result is the lengthening of rise time and fall time of keyed signals; in the same way, transmitters of voice or music have limited bandwidth allowance so that they must remove some of the high frequency sounds necessary for high fidelity broadcasting.)

This is not a failure of Fourier series, for the harmonic content of an electrocardiogram might not be a terribly useful bit of information in any event. However, if a formal description of a cardiogram wave is desired (and I doubt that anyone will contest this), then the description should be in terms of a few numbers, and the wave synthesized from these numbers should have properties indistinguishable under normal examination procedures from those of a true normal wave.

The instrument for describing a sharp spike is not the Fourier instrument (unless some application requires spectral analysis) but rather some instrument like the Bernstein polynomial. The Bernstein polynomial is usually defined over the interval [0,1] rather than $[0,2\pi]$, and I shall conform with this; undemanding arithmetic adjustments must be made to transform its effective domain to the interval $[0,2\pi]$. The Bernstein polynomials are indexed by two integers, n and v. The index n is always positive, and $0 \leq v \leq n$. The polynomial is defined as

$$B(n,v;x) = \binom{n}{v} x^v (1-x)^{n-v}$$

The quantity $\binom{n}{v} = n!/[v!(n-v)!]$ is a binomial coefficient used to normalize the polynomial. For fixed n and v, the graph of the polynomial attains a maximum at $x=v/n$. The function is monotone increasing to the left of this maximum, and monotone decreasing to the right; it remains positive on the open interval (0,1) and takes the value 0 at $x=0$

and x=1. As n increases, the sharpness of the peak increases. Professor T. S. Motzkin suggests that periodicity be restored by setting

$$x = \sin \frac{\theta}{2\pi} .$$

While there is no apparent theoretical reason for using what seems to be an unnatural mixture of polynomials and sinusoidal components to describe a function, there may be sound practical reasons for using the mixture to provide efficient descriptions of sufficient accuracy for calculation. In calculation there is frequently no demand for a complete nonredundant basis, or, perhaps, the knowledge that such a basis exists is sufficient to guarantee the soundness of a calculation.

The suggestion made above that Bernstein polynomials be used to remove troublesome peaks should not be interpreted to imply that all uses of Bernstein polynomials in computation are efficient. For example, the polynomials are a means of proving the Weierstrass approximation theorem that any continuous functions on [0,1] can be approximated arbitrarily well by a polynomial. For the function f, the n-th approximation is given by

$$f \sim \Sigma\, f(v/n)\ B(n,v)$$

The summation is over all values of v, starting with 0 and continuing through n. The use of the polynomials to prove the Weierstrass theorem is convenient. Roughly, the proof consists of showing that the method works for any quadratic function, noticing that the approximation is monotone in the sense that if $g \geq f$ for every point on [0,1], then the n-th approximation to g is nowhere smaller than the n-th approximation to f, and finally that any continuous function can be squeezed arbitrarily closely at any point by two quadratic functions, one of which is nowhere smaller than the squeezed function and the other is nowhere larger than the squeezed function.

Even though this is a convenient line of proof, there are usually more efficient ways of expressing a polynomial approximation to a function; these include the classical interpolation polynomials, which are easily computed; although they may not converge to the function approximated.

However, with good display and reasonably powerful means of computation, an on-line system might reveal the adequacy or inadequacy of a set of descriptors, and an ingenious operator might note peculiarities of the residue after a partial description, and might use these peculiarities to furnish progressing low redundancy descriptions.

I cannot stress too highly the value of such descriptors. Those of you who have dealt with the problem of recognizing patterns must recognize that labored descriptors, which are in some sense adequate from the point of view of a mathematical theorem, are inefficient and at least inadequate (more likely impossible) from a constructive computational point of view. With all this, however, we have few tools at hand to use in abandoning classical mathematical approaches toward description. The on-line systems now developing present a fine opportunity for augmenting our knowledge, or at least our practice, in this important field. Use of descriptors which are not orthogonal and do not form a linear set becomes difficult in classical functional analysis, but we can still use on-line displays from powerful computers to experiment. In short, then, one of the fields in which great interaction between formal manipulative mathematics and scholarly computation can be expected is the field of convenient description of functions or other mathematical entities. This description is classical through measure of interaction with each of a set of test entities, and our new power, if one is found, will lie in the enhancement of the allowable classes of testing entities, so that interaction between themselves, for example, can be tolerated more easily. An application of a low redundancy description might typically be construction of a Wiener-Hopf type of filter[7] to remove unwanted signals in a record, and to preserve the desirable ones. In some cases, it might be more convenient to use a small number of low redundancy descriptors abstractly than to subject large quantities of data to spectral analysis.

One field in which careful plotting and measuring might be substantially aided by good on-line displays is in the rational description of functions. This system of approximation was described and developed by Chebyshev, perhaps reintroduced by Hastings and his coworkers[8], augmented with tables by C. Lanczos[9], and applied in many laboratories with automatic computers. A recent presentation has become available through work sponsored by Control Data Corporation and participated in by Hans Maehly and many associates[10]. The set of approximations in this last reference is a fine set to consider in connection with on-line systems where the aproximations might be suitable.

I close this discussion of description by noting one additional development of our times which will certainly continue to interact strongly with on-line use of computers, and, for that matter, with development of computer languages for on-line or off-line use, and with more subtle uses of computers in the future. This is the development of quantitative studies of the conceptual aspects of all the communications sciences. This exciting field crosses many boundaries of classical factoring of scientific disciplines—biology, physics, engineering, psychology, linguistics, mathematics, and many separate divisions within these disciplines. One might profitably spend a few minutes considering what these newly expanding studies imply in connection with computation, and especially on-line computation. In this contemplation, the titles of the lectures of a series of lectures starting at UCLA might be illuminating. I present a list of titles as bibliographic references[11].

QUESTIONS OF STABILITY

It becomes clear that no one paper nor book can possibly describe the mathematics approriate for on-line computation. We all have recognized the growing awareness among appliers of mathematics to questions of stability. Perhaps one of the most valuable contributions made to mathematics by analogue equipment used during the last twenty-five years has been this growing awareness. Mathematicians have reviewed their work, and techniques have been discarded. Engineers have designed damping features with much greater care than was exercised in earlier times. The pace of current life has forced utilization of these studies of stability.

Where linear models are adequate, the analysis of stability is comparatively straightforward. However, as we become more and more dependent on models which are not adequately descriptive after linearization, these stability studies become more difficult and probably more important. It seems to me that we might expect on-line systems tailored for a class of stability studies as we develop greater needs for understanding the behavior of complicated non-linear reactors. The study which may develop into such systems is already being carried out in many different ways in many different places. I note the activities of the group called RIAS, first as a part of The Martin Company, in Baltimore, and more recently at Brown University. There under the leadership of Professor J. P. LaSalle and the inspiration of Professor S. Lefschetz, a productive group has developed many techniques which have not yet been widely applied to on-line analysis and simulation. At UCLA, other studies are progressing, and these are more or less typical of the type of thing developing at many places. They, too, tend to ferret out instabilities and near instabilities through computation, although the basic hope is usually to choose a technique which will be stable in the solution of some special problem in which instability is a nuisance, and not something to be studied for itself.

I look forward to further developments of our on-line systems to take into account the information now being accumulated and documented so that subtle, sensitive and more reliable simulations can be undertaken to determine behavior of complicated systems. I feel that on-line computation will contribute materially to this.

And, finally, I note that, except for physical control purposes, on-line computation has a kind of built-in suicide. As we learn more through its use, we must hope to formulate this knowledge, so that it is available without human judgement and intervention. In short, it must be included in the repertoire available for explicit off-line computing in much the same way as the development of D. H. Lehmer's technique, mentioned above, has enabled us to replace the isograph in solving polynomial equations.

ADDITIONAL TOPICS

I list here, mainly without reference, topics which now demand computational attention—most of it on-line.

In one-dimensional problems of the calculus of variations and in many similar problems, ordinary differential equations with end conditions arise. These may be solved by gradiant or relaxation methods, but such solution is frequently extremely laborious. On the other hand, they may be solved by initial value methods, creating a field of solutions with one to be chosen to pass through the second end point.

There seems to be some purpose in combining these two schemes. By on-line (or later, off-line) experimentation one might seek a well computed solution which passes close to the second end point. Then in some cases

perturbation methods applied to this solution through relaxation methods might give a very accurate solution with correct end values. I believe that experiments along these lines are planned at our Computing Facility by D. A. Pope and by James Dyer.

Similar studies might be applied to stochastic population process problems. These problems may yield partial differential equations or very large systems of ordinary differential equations. Some type of scaling experiments are well worth while in an effort to replace either system of equations by a set of ordinary equations small enough to be handled.

Another interesting study in stochastic population processes is the nature of configurations which are almost stable. In various attritive processes, a condition of almost stability occurs even though it is clear that complete annihilation of the population must eventually occur with probability one under the model used. Simulation of such situations seems attractive.

Other on-line simulations are desirable for the generation of models, for training purposes, and so on. The Link Trainers, introduced many years ago and improved since then, are famous examples of on-line computation somewhat specialized. Other training functions have been carried out spectacularly at the System Development Corporation. We still need models, systems, and simulation of highway traffic control and air traffic control. The UCLA Institute of Transportation and Traffic Engineering is carrying out interesting highway traffic studies, including many on-line simulations. The air traffic control studies are far less productive than one would have estimated ten years ago. They may have been hampered by poor models, on-line equipment inadequate for the purpose, or by other deterrent conditions. One thing which seems clear is the controlled traffic must be assigned to parametrized trajectory (the parameter being time) from take-off to touchdown. Thus, the awkward holding patterns used in the past (in which aircraft cruising at speeds in excess of a hundred knots pretend that they sit still to await their landing turn) must continue to be revised into long trajectory assignments terminating (hopefully) in routine landings.

Many of us would prefer that radical modifications of the approach and landing procedures be simulated first and then tried out on aircraft removed from our own position by some considerable distance.

Continuation of this list seems pointless. About the only thing to say is that simulation generally means that the result of some activity is estimated without actually going through the action. Newton's second law of motion is a simulation under this description—and I believe that it is best to accept this. Simulation is, of course, essential to the continuation of our ordered lives, and every plan is a result of some sort of simulation. Under these conditions, it seems clear that simulation of material activities will resume its place of exciting importance as the present trend of building up powerful on-line computing mathematical methods and computing and console response continues.

REFERENCES

[1] C. Tompkins, "Sperner's Lemma and Some Extensions," *Applied Combinatorial Mathematics*, Edwin F. Beckenbach (editor), John Wiley & Sons, New York, 1964, 416-455.

[2] R. L. Dietzold, "The Isograph, a Mechanical Root Finder," *Bell Laboratories Record, 16* (1937) 130-134.

[3] D. L. Lehmer, "A Machine for Solving Polynomial Equations," *Journal of ACM, 8* (1961) 151-162.

[4] T. S. Motzkin, "From Among *n* Conjugate Algebraic Integers, *n-1* Can be Approximately Given." *Bull American Math. Soc. 5* (1947), 156-162.

[5] *STL On-Line Computer* (volume 1, "General Description," by B. D. Fried, and volume 2, "Users' Manual," by C. C. Farrington and D. Pope) TRW Space Technology Laboratories, Los Angeles, 1964.

[6] R. H. Fox, "A Remarkable Simple Closed Curve," *Ann. of Math., 50* (1949) 264-5;
Richard H. Crowell and Ralph H. Fox, *Introduction to Knot Theory*, Ginn, Boston, 1963, p. 5.

[7] N. Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series*, MIT Technology Press and Wiley, New York, 1950.

[8] C. Hastings, *Approximations for Digital Computers*, Princeton University Press, Princeton, 1955.

[9] Cornelius Lanczos, *Tables of Chebyshev Polynomials*, volume 9, National Bureau of Standards Applied Mathematics Series, Washington, D. C., 1952.

[10] Control Data Corporation, A Study of Mathematical Approximations, 6600 Computer System Programming System Library Functions, Control Data Corporation, Minneapolis, 1964.

[11] University of California Extension Announcement, "Conceptual Bases and Applications of the Communications Sciences," (Spring, 1965); C. Tompkins, Introduction; Theodore H. Bullock, Neurons and the Simplest Transducers and Communication Channels; John L. Barnes, Laplace-Fourier Transforms and Mathematical Probability; Leo Breiman, Discrete Signaling and Coding Systems; E. Roy John, Electrophysiological Studies of Conditioned Responses; H. P. Edmundson, Mathematical and Computational Linguistics; Robert G. Gallager, Information Theory and Quantitative Measures of Information; Richard L. Masland, Manifestations of Structural Defects of the Central Nervous System; Joseph C. R. Licklider, Human Reception and Assimilation of Information; Ray L. Birdwhistell, Communication: A Continuous Multi-Channel Process; William J. McGill, Counting Processes in Psychophysics; W. K. Estes, Learning and Memory; David G. Hays, Representation of Meaning in Natural Languages; Gunther S. Stent, Storage and Translation of the Genetic Information; and Edward C. Carterette, Conclusion and Summary.

II

TECHNIQUES

Dr. Gene M. Amdahl*

Multi-Computers Applied to On-Line Systems

THE PRINCIPAL MOTIVATIONS for multiplicity of components functioning in an on-line system are to provide increased capacity or increased availability or both. The selection of a configuration should be based on sound economic valuation, just as the selection of the goals of the on-line system itself must be determined.

The use of multi-computers implies intercommunication, with the associated implications of interconnection, reconfiguration and interlocking. Some of the techniques, alternatives and effects of these implications are discussed.

CAPACITY

Although one of the principal reasons for multiplying components in a system is that of increased capacity, the efficacy of this procedure is not uniformly good. Generally multiplication for capacity is an economically desirable approach only for those components whose capacity is limited by technology. This is particularly true at present in electromechanical devices, such as peripheral storage and input/output (I/O) components. It is less true for electronic devices such as memory and central processing units (CPU's), in that order. In CPU's, adequate examples exist to demonstrate that doubling the component count more than doubles the performance, whereas duplexing the CPU less than doubles the performance. Doubling the cost of a memory would probably more than double its access rate, or certainly more than double its capacity, but probably not both simultaneously.

When multiplication is the most efficacious way to increase capacity, one can not expect a rate of capacity increase as great as the increase in components. The closest approach to a linear improvement can be achieved if the components can be partitioned, either for concurrent correlated use (such as in tape sorting) or concurrent independent use (such as spooling). The only factor limiting linear improvement in this circumstance is non-uniform loading, which almost always occurs. The second closest approach,and the only other technique known to this author, to linear improvement is by distributing, so that statistically a multiplicity of like functions can be performed concurrently. This can not always be done, for example, in printing, but can generally be done if extra-functional constraints do not make the component dedicated for extended time periods. To exploit the increase of capacity gained by distributing, the system must provide for queuing. The queue depth permissible should be at least as great as the number of components over which the function is distributed.

If more than one CPU resides in the system, another advantage of the multiplicity of other components appears. This advantage derives from the common pooling of equipment, even though the computer tasks may be independent. By pooling, the number of components provided need not be large enough to accommodate peak requirements occurring concurrently in each computer, but may instead accommodate a peak in one occurring at the

*Manager, Systems Architecture, IBM Data Systems Division

same time as an average requirement in the other.

Even though the capacity increase due to multiplication grows less rapidly than linearly, the hardware complement grows more rapidly than linearly. The additional hardware is required to provide the gating, selection, priority determination, and power needed to make the desired interconnection. This added circuitry not only increases hardware costs, but also reflects in increased failure rates and increased transmission delay. The increased transmission delay may be crucial in shared electronic memory, for the CPU-memory communication time is quite critical if the CPU is designed to attain maximum performance in combination with this memory.

A further precaution to observe when increasing capacity by multiplication is that even though functions may distribute very nicely in a statistical sense, not all devices are satisfied by good statistics. A prime example of this occurs in the use of shared electronic memories to accept or provide data to two or more high speed I/O devices concurrently. In this situation, even though the effective bandwidth is more than adequate to match the sum of the I/O device bandwidths, the interference, and consequent delays, may cause one or both devices to overrun. Overcoming this difficulty requires the inclusion of buffering to average the peak requirement to the effective bandwidth of the shared memory.

An alternative to multiplication is sometimes available, particularly in the CPU. This alternative consists of separating specialized functions from the CPU workload and providing a separate independent specialized component for this purpose. This approach is commonly taken in single computer systems, the most common example being I/O channels. This is also the essential point in those systems where a separate limited function, quite highly specialized, I/O control computer is introduced. In these particular systems, a performance gain can be greater than linear with respect to hardware increase. At present, there are not too many heavily employed specialized functions that are identified and substantiated, but this approach provides the greatest capacity gain payoff when it is applicable.

AVAILABILITY

The structure of a multi-computer system planned for high availability is principally determined by the permissible reconfiguration time and the ability to fail safely or softly. The multiplicity and modularity of system components should be chosen to provide the most economical realization of these requirements.

The system components involved in a given task form a configuration. The process of eliminating and introducing components when changing tasks is reconfiguration. The time required to reconfigure upon occurrence of a malfunction may be a critical system parameter. This reconfiguration time includes the time required for fault detection, fault location, switching, possible manual intervention, program restart, as well as supervisory program execution. If the lack of this faulty component requires performing the tasks in a different manner, new programs for the alternative procedures must be acquired.

The minimum reconfiguration time achievable appears in a duplexed pair of computer systems, each performing the task independently. If a component malfunctions, the reconfiguration time consists only of fault detection, fault location (to one of two computers), and possible switching of computer outputs to the on-line system. The supervisory task can be held to a minimum, and can even be implemented in hardware.

A multi-computer system which can perform the full set of tasks in the presence of a single malfunction is fail-safe. Such a system requires at least one more unit of each type of system component, with the inter-connection circuitry to permit it to replace any of its type in any configuration. More than one additional component is needed if a type serves as a repository for essential system information as well as other functions. Also, isolation must be provided, so that failure of one component cannot cause any other component to which it is attached to fail as well. The duplexed system described earlier is fail-safe.

A multi-computer system which can perform a satisfactory subset of its tasks in the presence of a malfunction is fail-soft. The set of tasks which must still be performed to provide a satisfactory though degraded level of operation, determines the minimum number of each component required after a failure of one of its type. Similarly the full set of tasks determines the maximum number of each component required, unless, of course, this number is not at least one greater than the minimum. Typically the fail-soft system consists of

two CPU's with a complement of I/O and storage. If the CPU's are unequal in power, the smaller must be capable of providing the necessary degraded performance. Any excess capacity in the fully operative system can be exploited by running diagnostic programs to detect potential or actual malfunction before production processing is affected. Significant advantages accrue if the CPU's are logically equivalent, even if not of equal performance. Such compatibility permits interchangeable programs and procedures which may multiply to accommodate to the various degraded configurations. Not only a smaller number of programs need be written, but also a smaller number need be retained in electronic memory, permitting more effective use of this always insufficiently provided commodity.

The degree of integration of the components used in the system may affect the availability as well as the cost of the system. Integration means combining two or more logically independent and separately interconnectable components so that common equipment is shared. The shareable equipment may be frames, power supplies, and even major portions of logic and data flow. It may even include common, but separately controllable, interconnection circuitry, if sources or destinations of data are common. The principal factors to consider when determining the degree of integration are the cost and circuit reductions achieved compared to the probable performance loss due to interference in the use of shared paths and vulnerability to the loss of both functional entities if common equipment fails. There are several reasons why the vulnerability to loss of both functions need not necessarily reduce the availability of the system. Firstly, if interconnection circuitry is common, the mating interconnection circuitry in the adjoining components is reduced equally. Secondly, the degraded system arrived at by failure of one function may not be able to exploit the second. Thirdly, the integrated components' functions may both be required for diagnosing the failing part. In practice, multi-computer systems are defined in which the integrated structures are significantly less costly and have somewhat greater availability with only a minimal loss of capacity arising from data path sharing.

COMMUNICATION AND INTERCONNECTION

So that the CPU's in a multi-computer system can function cooperatively, some communication between them is necessary. To be reasonably effective, two kinds of communication should be provided—a control signalling system which allows each CPU to gain the attention of the other, and a means of exchanging reasonably large quantities of information. The information exchange can be either by transmitting information between the CPU's over a connecting link, or by giving them access to a shared storage medium. The control signalling can be a separate connecting link or can be in common with the transmission link, if present. It should be able to initiate interruptions.

The transmission interconnection can be made for either local or remote CPU's in a system. For local transmission, a common control unit can be connected between channels of the CPU's, the control unit effectively making one channel appear to be sending information to a peripheral device and the other appearing to be receiving from a peripheral device. By such a tactic it is possible to achieve high speed transmission rates while utilizing existing system components for the major functions, and also permitting concurrent operation of both CPU's involved. For remote transmission a control unit is needed at both terminals of the transmission line with associated modulation-demodulation interfacing to provide proper operation. Since sending and receiving are essentially simultaneous, proper control signalling must preface transmission to establish rapport between the CPU's.

When two or more CPU's have access to a common storage medium, information placed in this medium by one CPU can be read by another CPU. In contrast to transmission, recording and retrieving are not simultaneous nor necessarily performed by different CPU's. Communication, differing in application and probably in implementation, is achieved by sharing of electronic memory, drums, disk files, or tape drives.

Shared electronic memory is useful for storage of common data and programs; in particular, the system supervisory programs and associated system status and component allocation data. Shared memory also permits distribution of tasks between CPU's on a more finely divisible basis, for the reconfiguration time for program switching of just the CPU is relatively very short. Shared drums and disk files are very useful for restart infor-

mation, permitting recovery on reconfiguration after malfunction. These shared peripheral storage devices also provide the capability for maintenance of a common program library which is useful not only for reconfiguration but also for the majority of programs normally required but not currently residing in electronic memory.

Shared storage media can appear in a number of configurations. All storage components of a given type may be shared, or some may be shared, while others are retained for private use by a CPU. For shared peripheral storage devices, the sharing interconnections may take place at several levels. Firstly, a pool of devices may be shared by several control units, permitting concurrent operation of a separate device by each control unit. Secondly, a group of control units may be shared by several channels, permitting concurrent operation of a separate control unit by each channel. Thirdly, the channels may be shared between CPU's, permitting partitioning of these facilities according to task requirements. The sharing of devices among control units is particularly important if a number of the devices become dedicated to particular roles in performance of a task. This applies to tape drives in sorting operations, and in this example the ability to share drives by pairs significantly improves the operation. For disk files and drums a single control unit frequently suffices for several devices, since each can play several roles within one task. Sharing of the control unit provides an effective sharing technique. If electronic memory is shared, the sharing of channels between CPU's need not necessarily involve any additional physical interconnection since the CPU desiring the channel function need not necessarily be the CPU which physically asks the channel to start in order to get equivalent action.

An additional point in shared electronic memory is that the commonly accessible portions should have the same addresses when referenced by different CPU's. By maintaining this convention, any CPU can pick up a partially completed task and begin execution from the current point. If addresses were not identical, the CPU would have to return to some earlier appropriate restart point and begin from there.

The interconnection circuitry which permits N out of M units to be operated concurrently by N controllers is an NXM crossbar switch. The crossbar switch may be built as a stand-alone unit, which then requires its own frame and power supply. It may also be distributed among either the controlled units or the controlling units, in which case it can be integrated to share frames and power. If distributed among controlled units, N points appear in each unit. If distributed among controlling units, M points appear in each. Usually there is no particularly strong reason to choose one over the other. For electronic memory, where transmission delays are very important, the distribution through memory units is more favorable, because priorities may be determined and put into effect slightly faster.

SOME DESIRABLE CHARACTERISTICS

In a multi-computer system designed for high availability, it is not sufficient to back up components with other components; it is also necessary to determine when and how this backup should take place. For this reason, extensive checking is very important. The design of the components should also be such that if they fail they do not cause malfunction of components with which they interface Such events as power failure introducing spurious pulses on the interface, or circuit failure holding an interface line at an inappropriate level, should be carefully investigated. These occurrences can make it almost impossible for the system to accomplish its own recovery.

If interrupts involve the use of defined memory locations, provision should be made to permit redefinition of these locations. If these locations are redefined, a CPU becomes independent of a specific memory unit for its operation.

Use of shared electronic memory permits several CPU's to alternately run supervisory programs. Since any one CPU does not know what any other CPU is doing at a particular instant, it is probable that two will vie for the same role at some time. So that chaos does not result, means for breaking such ties must be provided. The tie breaking can be performed by a reasonably simple program, but if this program is repeated too many places, its use is onerous. It is preferable to supply an instruction for the tie breaking facility required.

For a fully automatic system, it is imperative that the allocation of system resources be under control of a supervisory program. Pro-

vision for this control may be made by including the following characteristics:

1. A supervisory mode with associated privileged instructions which must be used to allocate a resource initially, but not necessarily control its use after allocation.
2. Storage protection to ensure survival of the supervisory program and its resource allocation data, with storage protection changes performed only by privileged instructions.
3. Hardware monitoring of changes in system component status with the ability to return the CPU control to the supervisory program by interrupting current processing.
4. An interval timer to return control periodically to the supervisory program, so it can take stock of the system status.
5. A wait state, sensitive to interrupts, available to the supervisory program, rather than a stop or halt instruction available to other programs.

By means of these provisions, a multi-computer system can be made capable of sustaining an on-line status in spite of a wide variety of possible circumstances which can arise in practice.

SUMMARY

The use of multi-computers in on-line systems is clearly of growing importance as these new applications continue to be formulated and developed. The multiplication of components in these on-line systems is not a desirable end in itself, for their usage efficiency per unit drops and many new problems are introduced. This multiplication does, however, provide for capacity gains not otherwise achievable and for system availability of very high level. Without these gains in sight, the useful on-line system applications would be quite restricted.

On-line application itself implies the need for certain system characteristics. Such characteristics provide the ability to remain responsive by controlling the marshalling of system resources to best meet the current needs. Such marshalling requires the ability to reconfigure to exclude malfunctioning units or to reallocate assigned units when necessary.

ACKNOWLEDGEMENT

The author is indebted to Dr. G. A. Blaauw for organizing and providing much of the material presented in this paper. His published article on a similar topic with respect to IBM System/360 is recommended for reference—G. A. Blaauw, The Structure of System/360, Part V, "Multisystem Organization", *IBM Systems Journal*, Volume Three, Numbers Two and Three, 181-195, 1964.

Joseph Weizenbaum*

On-Line User Languages†

THE TITLE suggests that there is some difference between on-line and off-line user languages. The fact that such a paper was invited at all at this time suggests further that there is a renewed interest in that distinction. I say "renewed" because our experience with computer languages began with on-line systems. Those of us who are now privileged to have on-line access to large scale computers often have a distinct *déja vu* feeling; we have been there before. Of course we have. It was in those far away days when the only way to communicate with computers was by the direct coupled input/output devices euphemistically called "consoles", often consisting of arrays of switches and buttons of such bewildering complexity as to provide a challenge to a modern airplane pilot. A little later, some computers were designed by people who had actually used earlier models themselves. That accounts for the appearance of typewriters on some early machines. Many of the modern small machines are direct descendants of these early models, differing from them only in that they have core memories in place of drums or mercury delay lines, and in that they are very much cheaper. The early machines were operated in an on-line mode because techniques for efficient batch processing were not yet developed, today's small machines are operated that way because their economics are not prohibitive. On the contrary, most large scale computer systems are *not* operated in an on-line mode because the economics of such operation *are* prohibitive. I shall return to the economic question below. For the moment, I wish merely to implant the idea that, were it not for economic constraints, most computer users—certainly most programmers—would prefer to deal with a computer on-line.

It is sometimes suggested that a very large fraction of computer time is used in running programs merely to find out why they do not work. Whatever the statistics might be, it is certainly true that debugging of programs is one of the chief preoccupations of computer users. But it ought to be remembered that a program is somebody's strategy for solving some problem, no matter whether that program has bugs in it or not. The debugging of a program is another problem. It is such a common one that sets of techniques for attacking it emerge out of the common experience.

What is really going on when a programmer is engaged in debugging his code? In effect he is doing science. By this I mean that he is interrogating some ill understood aspect of nature. He forms hypotheses based on his current understanding, designs and applies tests for verifying them, intellectually analyzes the outcomes of his experiments, modifies his hypotheses, and so on. When he finally understands the nature of a particular bug, he modifies his code to remove the difficulty. The crucial difference between this kind of activity and that of writing programs is that debugging is an exploration of a solution space with the aid of a computer, while the latter is merely the encoding of solutions already arrived at by other means! We all know how absolutely necessary the computer is to the debugging process. It is difficult to the point of impossibility to diagnose an ailing program by pure reason alone. This then

†The preparation of this paper was sponsored in part by the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government.

*Associate Professor, Dept. of Electrical Engineering, Massachusetts Institute of Technology.

explains why the first language systems which can truly be classified as "on-line" were those, like DDT for PDP-1, designed as debugging aids.

What I am really trying to emphasize by this argument is the principal operational advantage to being on-line with a computer; namely, that that mode of operation permits the computer aided exploration of solution spaces, as opposed to the mere exercise of programs representing solutions already encoded. It seems to me that the facilitation of this exploratory use of computers is the entire aim and direction of all work in computer language development, and of all computer systems design not motivated by pure data reduction requirements. It follows then, that on-line computer languages should be looked upon from this point of view, and judged on the basis of criteria derived accordingly.

On-line access to large scale computer systems has now been made possible as a consequence of the development of sophisticated time sharing systems, such as the ones of which Project MAC, at the Massachusetts Institute of Technology, is an example. While it is no doubt appropriate to list the various on-line languages which have been developed within Project MAC, and to catalogue their characteristics, it is perhaps more useful to first call attention to an important difference between simply having on-line access to a large computer system and having on-line access to a large time shared computer system. A user having his own IBM 7094 has access to whatever software happens to come with his machine plus all he manages to add to the library as a consequence of his own efforts. A time shared system, on the other hand, is constantly enriched by the combined efforts of *all* its participants. Our experience with Project MAC underlines the importance of this distinction with very great force. Of course, an extensive executive system is required to make programs developed by anyone available to everyone, and to honor constraints imposed by program and file protection conventions at the same time. However, the operation of a time shared computer system demands a complex executive program in any case; not much more is needed to meet this somewhat expanded goal.

The above observations are relevant to the discussion of on-line user languages in that the various system commands which are, so to speak, at the fingertips of the user of the time shared system, constitute a kind of on-line language in themselves. As it happens, no one has taken the set of such commands within Project MAC and codified them in, say, Backus Normal Form. That may, in fact, not be possible. If it is not, then this must be because this set is very much *ad hoc*. But, lest this be taken as a criticism, let me add quickly that it follows from the very way in which such commands are added that no strict advance plans can be made. It is not possible to predict, after all, when some user may come up with something of general interest and utility.

For the present purpose, we may look upon the time sharing executive as an elaborate interpretive program. Most interpreters have some mechanism for getting at the "next" instruction. There may be a pseudo program counter, for example. The time sharing executive gets its "next" instruction from the input buffer associated with a currently active program whenever that program has ascended, so to speak, to the system level. The set of instructions which it will understand, and to which it will respond, constitutes the set of "system commands" alluded to above. The "interpreter" itself imposes some syntactic constraints on the sequences of commands it will accept (e.g. certain sequences of commands are ungrammatical). The view of the time sharing executive as an on-line language structure is reinforced when it is noted that it is possible (within the MAC system) to file chains of system commands, i.e., in effect, to write a program in the vocabulary provided by the set of system commands, such that when that program is subsequently executed, the effect is as if each command were typed in separately and in sequence after the previous command has been obeyed. One way of making a new program, i.e., one written by one of the MAC users, publicly available to all MAC users, is to modify the interpreter to enable it to respond to a new instruction, namely, that which loads that new program into core when called upon, and prepares it for execution.

There are, however, certain languages within the MAC system which may be characterized as "on-line languages" in a more conventional sense. These all share the property that they are interpreters. (I use the word "interpreter" somewhat guardedly for I do

not believe that a very precise distinction between interpretation and compilation can be made—nor do I believe that such dichotomies are useful.) Clearly an on-line language within a time sharing system should have the property that its user be able to engage the machine in more or less intimate conversation; i.e., to exchange messages with it on a give and take basis and with human tolerable frequency. This means that the user will write generally very short programs leading to intermediate results, on the basis of which he then decides on his next course of action. It is fairly obvious, and indeed it proves to be so in practice, that this mode of operating a program contributes greatly to the kind of exploratory use of the computer mentioned earlier. An important consequence of such operating practice is that the programmer does not have to anticipate every possible eventuality and account for it somehow in his program (even if only to provide an error exit).

Any already existing interpretive system (e.g. LISP), which normally expects its inputs in the form of sequences of cards, and delivers its outputs to either tape or printers, can easily be modified to look for its inputs from the typewriter console, and deliver its outputs to the same instrument. One such programming system is an interpretive version of SLIP augmented by arithmetic and control functions. This system is called OPL (*O*n-Line *P*rogramming *L*anguage). Up to a point, OPL's architecture is a prototype for a number of other such systems operational in MAC.

The basic input to OPL is a list structure. Generally speaking, an input list structure contains a number of expressions in functional notation; for example, "LOG(SQRT-(X))", separated by "$". All the functional operators, except the four basic arithmetic operators and "=", are prefix operators; the exceptions are infix operators. Since, in more or less standard list notation, a "(" denotes the beginning of a list or sublist, and a ")" the end of one such, the input list structure is already in the form of a tree by the time it needs to be interpreted by the OPL machinery. OPL is itself written in SLIP, and SLIP list processing operators are used to administer the entire interpretive process. This means that a large part of the SLIP library must be in core during the interpretive cycle. Since this is necessary in any case, the same SLIP functions used by OPL internally may as well be made available to the OPL user explicitly. OPL, therefore, contains a table (essentially a large transfer vector) which associates with the name of each available SLIP function the entry point to that function. The same table similarly gives the entry points to the non-SLIP functions which round out the OPL system.

The final result is a quite general purpose programming language of a power roughly equivalent to that of LISP. It is, however, considerably more mnemonic than the latter. Since it was designed to be an on-line language, it has certain features which merely transformed interpreters, arising out of different motivations, do not (but could easily be made to) have. One example of such a feature is that a previously undefined function may be called in a program (no matter how deeply nested). When the call is encountered, a message is typed out that an undefined operation has been encountered, and the program held in abeyance until the programmer enters a program defining that function. From then on (and for the purposes of that program) the newly defined function behaves just as any other built-in or previously defined function.

OPL programs and their associated data structures may, of course, be stored on the disk. The OPL user may, therefore, build up very complex data structures over a long period of time, experiment with them on line whenever he wishes, and freeze them in intermediate states for subsequent exploration. A simulation of the effects of various organizations of a business firm may, for example, start with a quite simply developed tree representation of a small firm. Programs which compute differing budgeting strategies for such a firm may be exercised on line, and the most interesting ones saved on the disk file. As experience with the behavior of the model accumulates, the firm can gradually be grown both in size and complexity. The model maker may sit at his typewriter for hours, pushing the model around, modifying it, testing its sensitivity to this or that. In the process he is, of course, spending most of his time thinking—in fact he is using (and being charged for) very little machine time. At some point he will store his program in its then current state to continue manipulating it a few hours, days, or even weeks later.

OPL is, as I pointed out, a general purpose on-line language. There exist a number of special purpose languages within MAC which have roughly the same structure as OPL. One of these is COGO (Coordinate Geometry Program) which was developed and is used by the M.I.T. Civil Engineering Department. It is similar to OPL in that it too uses essentially the same transfer vector philosophy. The functions which may be called have all been previously defined and compiled in either FORTRAN or MAD. COGO is used by students in the Civil Engineering Department for laying out highway interchanges and like purposes. The programmer defines points in a two dimensional space and asks for them to be connected by various curves and straight lines. He then interrogates the system about the consequences of such connections, e.g., the areas determined by various enclosed surfaces. An important property of COGO and other programs in the same class is that the student needs to know literally nothing about the structure of the COGO program itself nor about programming in general. STRESS is a similar program which deals with the stress analysis of structures under loads.

The principal reason the users of these programs need know nothing about programming is that both COGO and STRESS are essentially self teaching. By this I mean that they not only respond to the stimuli provided by their users in the sense of yielding intermediate results, but give directions for their proper use. A particularly strong example of such a program is one developed by Hansen and Pyle for the design of nuclear reactors. This program not only asks for its relevant parameters by name e.g. "What fuel do you intend to use?"—but also comments on the relative appropriateness of the response under certain condition – e.g. "That's rather large"—and gives the user a chance to change his mind before going into its calculation phase.

OPL, COGO, and STRESS are each programs which, while being on-line, are still conventional in the sense that their users specify procedures in terms which are generally recognizable as program steps and produce results which other programs might well also produce. A radically different class of on-line programs is represented by the "ed" and "typset" programs available within the MAC system. These are both editing programs. They differ from one another mainly because the "ed" is designed to process computer code (in whatever language), while "typset" is a general text editor. I will take up only the latter.

Typset operates in either the "input" or the "edit" mode. In the former the user types in whatever text he pleases, using all characters available on either the IBM 1050 or the Teletype keyboard (with the exception of two escape characters, one of which is used to delete an entire line and the other a single character—both may be changed at will). In the edit mode the text is edited entirely by context. There are no concepts such as line numbers nor any others requiring the user to remember the appearance of the original text. In the edit mode, paragraphs may be interchanged, new text inserted between pairs of words or characters, strings of characters deleted, and so on. Text which has been input via the typset program is ultimately stored on the disk and may be output on any kind of paper (e.g. ditto masters) at any time. Since it does reside on the disk until intentionally deleted therefrom, it may be re-edited repeatedly days, weeks, or months after its original composition.

It seems to me that the "typset" and "ed" programs are significant for two separate reasons. One is that they provide examples of programs which anyone may use to very good advantage—anyone, regardless of how little or much computer programming he has behind him. Even I have been able to turn out papers with lines properly centered, left and right margins justified, and the proper spacing between paragraphs. Perhaps of more immediate importance is the fact that both these programs were written, debugged, and put into general use within a few weeks of their inception. This, while of credit to their authors, certainly also serves as a comment on the utility of a time sharing system and the on-line program writing and debugging facilities it provides to each of its users.

Now to return to the question of economics which I postponed earlier. As already indicated, I believe the economic issues related to the on-line use of computers must be separated according to whether one is speaking of small computers operated from their directly connected consoles or large time shared computer systems. The more challenging and interesting questions certainly lie in the latter

category. Visitors to Project MAC are constantly asking how much machine time a problem they solved on a batch processing system would require on the MAC system to come to the same state of solution. I suppose they want the answer in terms of numbers of machine cycles or some related measure. Perhaps these questions can be answered with precision in some cases. Whether such answers would be impressive would depend, of course, on the efficiency of the batch processing system with which the MAC system is being compared. Batch processing also entails overhead. However, I do not believe such questions to be very meaningful—no more so, for example, than asking for the average floating point add time of a machine like the STRETCH leads to any insight into the overall effectiveness of such a machine in relation to specific problem classes. It must be remembered that economics deals with the allocation of scarce resources. To consider the number of machine cycles required to solve a particular problem to be the principal measure of effectiveness of a system assumes that the scarce resource to be conserved is machine time. It is not. In any case, one would probably not take the same code as that which was generated for the batch processor and run it in the time shared environment. In the batch processing environment the programmer knew that access to the machine was limited to a few shots a day. He wrote his code in anticipation of every conceivable event, asked for large dumps, interleaved the operationally significant portions of his program with elaborate diagnostics, and so on. None of these precautionary measures are necessary with on-line operation. Programs are therefore shorter and run correspondingly faster.

The scarce resource which is being conserved in the on-line mode of computer operation is the energy and time of people. Our experience with Project MAC has taught us that the exploratory use of computers, such as has by now become habitual with us, serves to amplify the effectiveness of people in dramatically visible ways. Just as the introduction of the computer itself made possible attacks on problems which simply could not be attempted earlier, so we find that our mode of operation encourages people to search their own problems more deeply, to be dissatisfied with sloppy solutions which might have passed earlier because "after all, they work". In short, we are beginning to see the use of the computer as a real and significant assistant to human beings engaged in problem solving. Who is to say what economic values accrue to an institution when creative people complete their tasks in days instead of weeks, or when a problem which could previously not be attacked at all is now solved?

ACKNOWLEDGEMENT

Work reported herein was supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research contract NONR-4102(01).

III

APPROACHES

Werner L. Frank*

On-Line CRT Displays: User Technology and Software

INTRODUCTION

THE INCREASING APPLICATION of on-line data processing systems has accelerated interest in visual display devices for augmenting the interaction of man and machine. To date, this technology had been primarily developed for the military, where the requirements of air defense stimulated the first application of individual display console devices for the Semi-Automatic Ground Environment (SAGE) and the Naval Tactical Data System (NTDS).

Visual displays have been with us from the earliest days of computing. For example, one of the first display devices associated with a computer system was the cathode-ray tube (CRT) display. One of these displays was available in 1953 on the ILLIAC (University of Illinois) computer where two tubes were driven in parallel. One CRT was mounted for visual observation, the second was associated with a camera capable of photographing the computer generated display. The computer controlled the film advance. While the primary use of this display was the rapid generation of graphic information, another use was on-line monitoring of the progress of a calculation. By appropriate displays, a programmer could detect programming errors, or, during production runs, make better initial guesses for iterative procedures or parameter studies. Subsequently, such systems became available commercially as, for example, peripheral devices to the IBM 704 computer system.

*Vice President, Eastern Operations, Informatics Inc.

This paper explores the current state of the art of CRT displays and discusses, from the user's point of view, potential applications and the computer programming software necessary to implement these systems.

CRT DISPLAY CONSOLE CHARACTERISTICS

From many points of view, all of the following are on-line display devices: typewriter, plotter, printer, closed circuit TV, document viewers, CRT consoles, and tote boards.

This paper discusses the CRT display console which meets three on-line capability criteria. First, it is directly tieable to a data processing system. Second, it has ability to initiate messages or control signals from a data entry keyboard or switches for transmission to the computer. Finally it has ability to receive digital messages or control signals from the computer and display them to the operator or viewer.

Typical Features

The CRT display console is a desk type unit with varying combinations of features and capabilities for information entry and display. The primary entry devices are the alphanumeric keyboard and switch action keys. The output is typically produced via a CRT and status indicators. Aids to data entry and output observations are provided. **Table 1** summarizes these features, and includes associated check points which allow a specific device to be evaluated. **Figure 1** is a conceptual drawing of a display console identifying these generic features. An important distinction is made between the marker, light pen

and cursor, each representing a unique capability for data entry.

The marker defines a position on the face of the CRT where a character appears when selected from the keyboard. This advances across the face of the display as characters are successively entered.

The light pen is a photoelectric device which can be aimed at any position on the CRT. When the light pen is activated, and the point on the CRT at which it is aimed emits light, a message is generated to the computer identifying the location of that point.

The cursor is a feature which permits the location of any point on the CRT, independent of the presence of light.

These three capabilities can be satisfied under software control and in conjunction with several variable function keys. It is also possible to meet these requirements with hardware features in several ways. For example, the "position-pencil control" of the IBM 7460 Special Image Processing System encompasses the features of both the light pen and cursor.

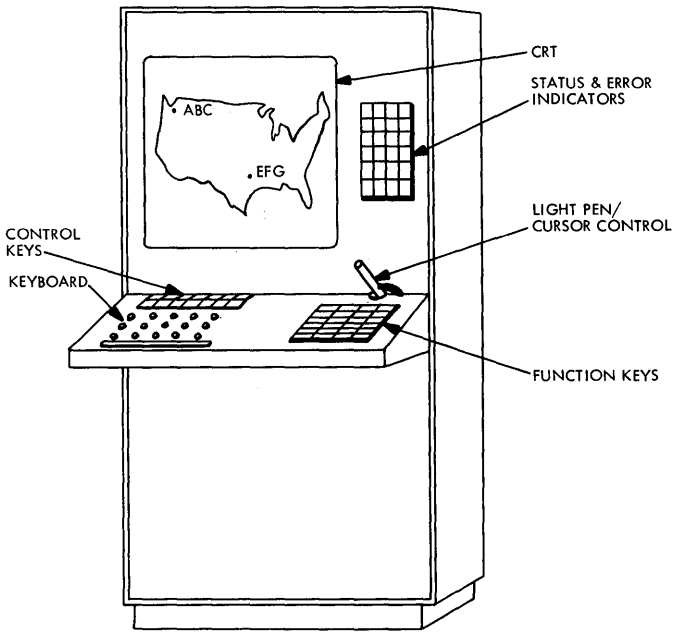


FIGURE 1

TYPICAL CRT DISPLAY CONSOLE

TABLE 1

CRT DISPLAY CONSOLE FEATURES

	Check Points	Associated Features for Data Entry	Associated Features for Output Observation
Alphanumeric Keyboard	Number of characters Availability of special Symbols	Marker indicating on CRT point of character entry	
Switch Action Keys	Number of keys under program control (Variable Function Keys) Number of keys under hardware control (Fixed Function Keys)	Lights associated with keys for indicating: - key selected - allowable key selection	
Cathode Ray Tube	Size of screen Number of characters that can be displayed Position accuracy Ability to draw vectors Number of character sizes	Light pen for point selection on CRT Cursor for locating position	Selected output features: -blinking of characters -intensity variation -character size variation -character rotations Mix analog and/or video information with computer generated digital information Color presentation
Status Indicators	Number of lights under program control Number of lights under hardware control		Ability to blink light Ability to control audible alarm

A summary of a number of commercially available display devices and features is given in **Table 2.**

Operations

The utility of the features associated with on-line display consoles is measured in terms of each application. It is possible, however, to define operations common to many applications and assess the need for the capabilities identified in **Table 2**. Such a list of common operations includes: initiate an action or program in the system; send a message to one or more other console stations or the computer; request a hard copy output product; view an output generated for the CRT display; perform a logical operation at the console by a man/machine oriented procedure (e.g., data base query or special computations); control access and viewing of background projections; and generate visual displays for storage and later viewing, specifically with reference to background projections.

The console hardware capabilities or features required for performance of these operator functions are shown in **Table 3.**

Consider the first item. To initiate an action, a key must be pressed. This is usually one of the variable function keys. For illustration, let such a key be labeled "Mode Selection." The pressing of this key may cause, under program control, the display of a list of alternatives as shown in **Figure 2**. The desired mode is then selected by pointing the light pen at the appropriate asterisk, say opposite HARDWARE SYSTEM DIAGNOS-

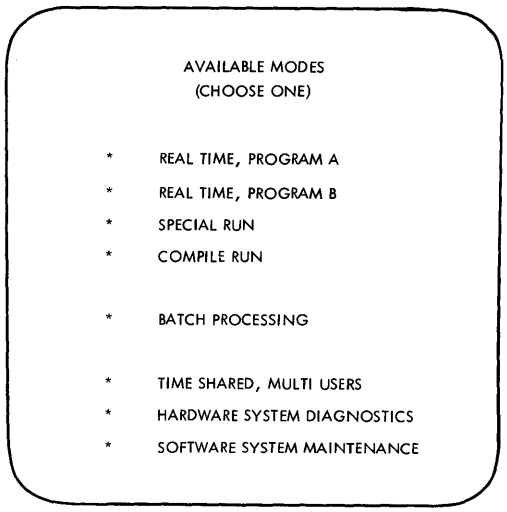


FIGURE 2

LIST OF CHOICES: "MODE SELECTION"

TABLE 2

SUMMARY OF ON-LINE CRT DISPLAY CONSOLE CHARACTERISTICS

	A/N Display	Vector Drawing	Variable size characters or fonts	Light Pen	Marker	Cursor	Buffer Memory	Fixed Function Keys	Variable Function Keys	Status Indicators	Blinking	Audible Alarm	Background Projection	Character Rotation	A/N Keyboard Input	Selective Intensity Variation	Cost in $1,000	Remarks
Typewriter (for comparison)	x		x		x										x		3.5	
Data Displays: DD10	x				x			x							x		5.2	Up to 64 displays per control unit at $25K extra
Data Display: DD80	x	x	x	x	x	x	x		x					x	x	x	125	Has associated film recording capability
General Dynamics: SC 1090	x	x						x		x							40	Additional options available
Bunker Ramo: BR85	x	x	x	x	x	x	x	x	x	x	x	x			x		160	
Raytheon: DIDS500	x		x	x	x		x		x	x					x		?	Joystick provides equivalent of light pen
ITT: Information Display Console	x	x			x		x	x	x	x		x	x		x		200	Includes color presentation
IBM: 2550	x	x	x	x	x				x						x		75	Buffer memory optional

TABLE 3

RELATIONSHIP OF CONSOLE FEATURES TO GENERIC MAN/MACHINE OPERATIONS

Function	Console Features										
	Alpha-Numeric Display Capability	Symbols	Cursor	Light Pen	Marker	Keyboard (Alpha-Numeric)	Fixed Function Keys	Variable Function Keys	Line Drawing	Blinking or other Alarms	Background Projection
Initiate Action of Program	x			x	x	x		x			
Send a Message to Other Consoles or to the Computer	x				x	x	x*	x			
Request a Hard Copy Output Product	x				x	x	x*	x			
View on Output	x	x							x	x	x
Perform a Logical Operation	x			x	x	x		x			
Control Background Projection	x		x or	x	x	x	x	x			
Generate Visual Displays for Storage	x	x	x or	x	x	x	x*	x	x		x

*Can be implemented hardware or software

TICS. It may be that the item selected requires further detailing. Hence, a second display automatically appears forcing the operator to make a further choice, as in **Figure 3**. Actions for manipulating the "list" display require only the variable function keys, an alphanumeric display capability, and the light pen for selection.

The selection of CORE MEMORY from the list in **Figure 3** may force the format of **Figure 4** on the CRT, requiring the indicated input from the operator. The symbol " " represents the "marker" and indicates the first entry point when the alphanumeric keyboard is used for data entry. This marker moves either under hardware or software control to the next succeeding underline as each character is entered.

For this latter action of handling "format" displays, the marker and data entry keyboard is required, completing the explanation of features needed to accomplish the first function of **Table 3**.

APPLICATIONS

The on-line CRT display console is rapidly reaching a degree of acceptance in the commercial environment as evidenced by the availability of display devices as standard peripherals to many of the newly announced computer systems. The development of application areas and user techniques has, however, lagged so

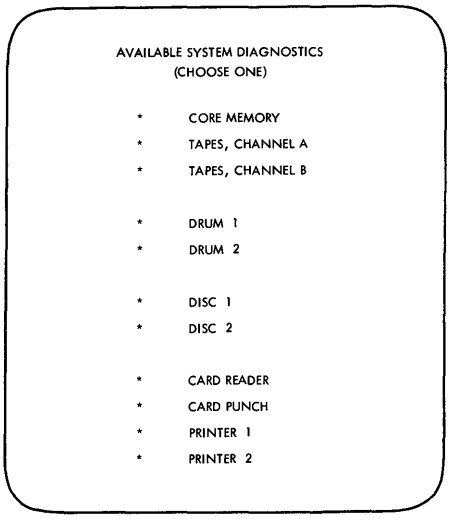


FIGURE 3

LIST OF CHOICES: "SYSTEM DIAGNOSTICS"

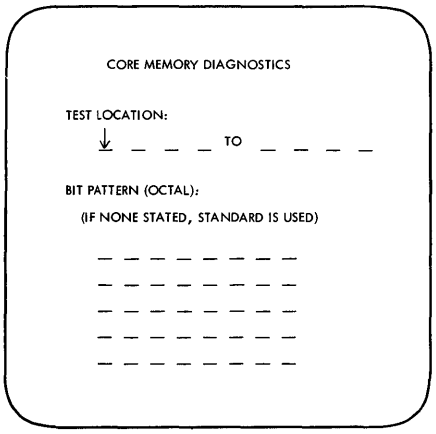


FIGURE 4

FORMAT FOR ENTERING REQUESTED INFORMATION

that there is still considerable potential user caution and hesitancy.

Three major application areas have developed: demand query, monitoring, and analysis. In the first, a service is provided to a person who wishes immediate information that must be extracted from a large mass of data and may require some rudimentary analysis or transformation before it is useful to him. Examples are: a manufacturing planner wants to know the status of a release order for certain parts; an investor wants to know the status of his account with his broker; and a policy holder wants to know how much he can borrow on his life insurance policy.

Included as candidate applications for demand query are: reservations (travel, hotel, etc.), merchandising inventory, manufacturing inventory, manufacturing and production control, insurance policy information, credit information, bank or brokerage customers account information, real estate information,

TABLE 4

DISPLAY CONSOLE OPERATING CHARACTERISTICS FOR SELECTED APPLICATIONS

Applications	Characteristics: General					Output						Input			Man/Computer Interaction			
	Single Station	Few Stations (2 to 10)	Many Stations (10 to 1000)	Small Qty. (10 to 100 char.)	Large Qty. (100 to 1000 char.)	Numeric Only	Alphanumeric	Vector Drawing	Hard Copy	Read Out Only on Demand	Continuous Monitoring	Highly Encoded	Arbitrary Messages	Operating Pointing	Slow Interaction	Fast Interaction	Analysis Done Mostly By Man	Anal. Done Mostly by Computer
QUERY																		
Reservations (travel, hotel, etc.)				x			x			x		x			x		x	
Merchandising Inventory			x	x			x		x	x		x			x		x	
Manufacturing Inventory		x		x			x		x	x		x			x		x	
Manufacturing and Production Control		x			x		x		x	x		x			x		x	
Insurance Policy Information		x		x			x		x	x		x	x		x		x	x
Credit Information		x		x		x				x		x			x		x	x
Bank or Brokerage Customers Acct.Infor.		x		x		x			x	x		x			x		x	x
Real Estate Information		x			x		x		x	x		x			x		x	
Stock Quotation			x	x			x		x	x		x			x		x	
Management Control	x				x		x	x	x	x		x		x	x		x	x
Computer System Status	x				x		x			x		x				x	x	x
Patent or Law Precedent Search		x			x		x		x	x			x		x		x	
MONITOR																		
Computer-Monitored Tests, Process Control	x				x		x	x	x		x	x		x		x		x
Hospital Operations		x			x		x	x	x		x	x				x		x
Simulation	x				x		x	x			x		x	x		x		x
Stock Status			x	x			x				x	x				x	x	
ANALYSIS																		
Information Retrieval		x			x		x	x	x	x			x	x	x			x
Input Data Screening and Updating		x			x		x		x	x		x		x	x		x	
Tape File Editing	x				x		x		x	x		x			x		x	
Remote On-Line Programming or Debugging		x			x		x	x	x	x			x	x	x		x	x
Electro Computer Aided Drafting		x			x		x	x		x			x	x	x		x	x
Teaching Machines			x		x		x	x		x			x	x	x		x	x
War-Gaming					x		x	x			x		x	x	x		x	x

stock quotation, management control, computer system status, and patent or law precedent search.

For a demand query, the user is interacting with a display in a very active manner. The user has an objective in mind. He makes an inquiry of a general nature; a display is presented. Based upon this newly acquired information, the user may particularize his inquiry. A new presentation gives him additional information. The sequence continues until the user has all the information required to take some action.

For display monitoring, the user does not interact at all with the display. He is entirely passive. The presentation made by the display may change at a slow or rapid rate; the user observes it and obtains the information necessary for action. Typical applications include: computer-monitored tests, process control, stock status, hospital operation, and simulation.

In the third application area, the display console is used as an analytical tool to fulfill the basic task itself. Here the man and machine operate in conjunction. Examples include: information retrieval, input data screening and updating, tape file editing, on-line programming or debugging via remote console, computer aided drafting, wargaming, and teaching machines.

The foregoing applications may be analyzed in terms of certain characteristics. First, the number of stations required. For example, airline reservations require many stations, each agent having an individual set. On the other hand, the director of a computer monitored test requires only one station. Next, input and output can differ completely from one application to another with respect to quantity, degree of encoding, number of symbols, whether it is pictorial, etc. Also, the degree of interaction between the user and his (computer-controlled) display differs in speed of required response and in the way that the responsibility for analysis is divided between the man and the computer.

Some characteristics for the applications listed above are shown in **Table 4**. General conclusions for each class of applications are given in **Table 5**.

APPLICATION EXAMPLE: QUERY

One of the more important uses of the on-line console is to ask questions of the data processing system. This process can be as simple as pressing a key and inserting an appropriate five character code to obtain a bank balance, as with demand query. Or it may be a complicated logical statement whose rules of formulation may be as complicated as

TABLE 5

SUMMARY OF OPERATING CHARACTERISTICS FOR SELECTED APPLICATIONS

Application Class	Number of Stations	Output	Input	Man/Computer Interaction
Demand Query	Many	A/N; Varying Amount	Highly Encoded	Slow; Analysis by Man
Monitor	Few	A/N & Vector; Mostly large Quantity	Encoded	Fast; Analysis by Machine
Analysis	Few	A/N & Vector; Large Quantity	Mixed; Coded & Message	Slow; Analysis divided by Man and Machine

the search process which obtains the answer, as in interrogation for information retrieval.

Consider, for example, the conventional off-line procedure for interrogation as shown in **Figure 5** . Here, the user may be as much as nine steps removed from the solution of his problem. Beyond this limitation the conventional process would require, at some stage of operation, adherence to very formal rules. The first of these rules is concerned with spelling—abbreviations, plurals, possessives, may be illegal. Next is the use of special words—synonyms may be prohibited, special codes may be required. Then there is punctuation—queries may have to be marked off and segmented. In short, a special syntax, dictionary and code book may be required for use of the system.

The on-line display console facilitates the process of query by eliminating the intermediaries controlling input (and hence errors) and automates the dictionary and syntax rules.

Consider, for example, an on-line system which permits query of a data base consisting of information about the stock market. We assume that a satisfactory set of variable function keys consists of keys for: MARKETS, INDEXES, INDUSTRY CLASSES, SUMMARY LIST, SELECTION CRITERIA, DETAIL LIST, GEOGRAPHIC, PLOT, and TIME.

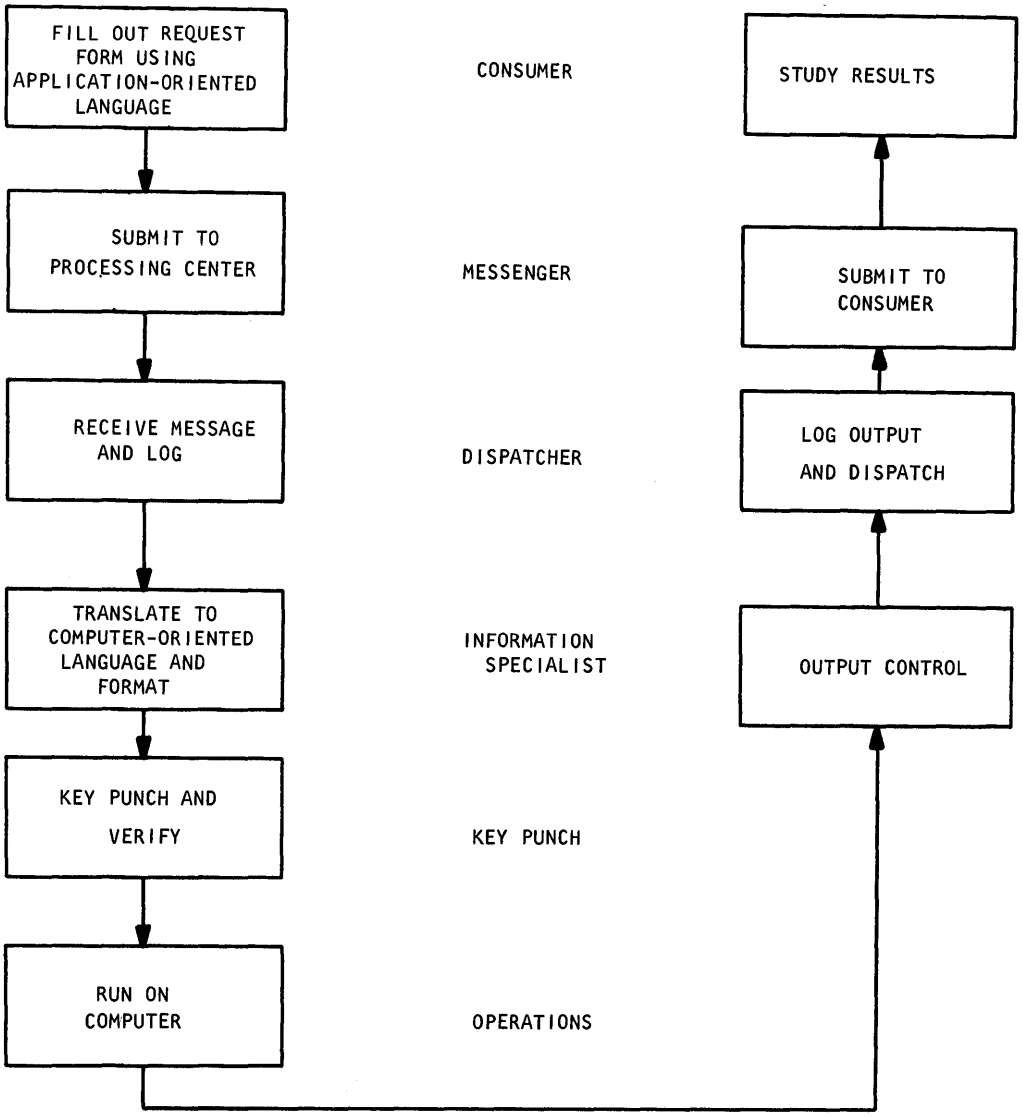


FIGURE 5

CONVENTIONAL PROCEDURE IN REQUEST FULFILLMENT

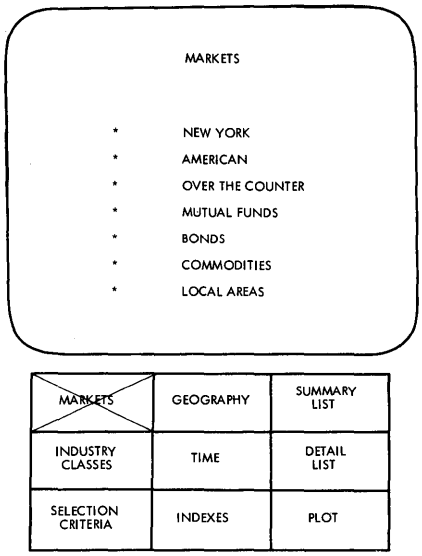


FIGURE 6
LIST DISPLAY OF MARKETS

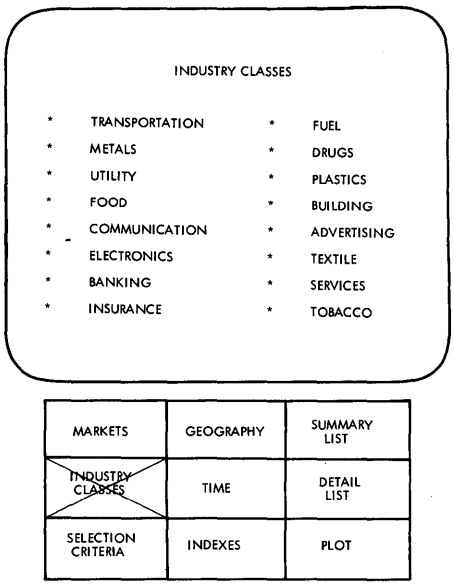


FIGURE 7
LIST DISPLAY OF INDUSTRY CLASSES

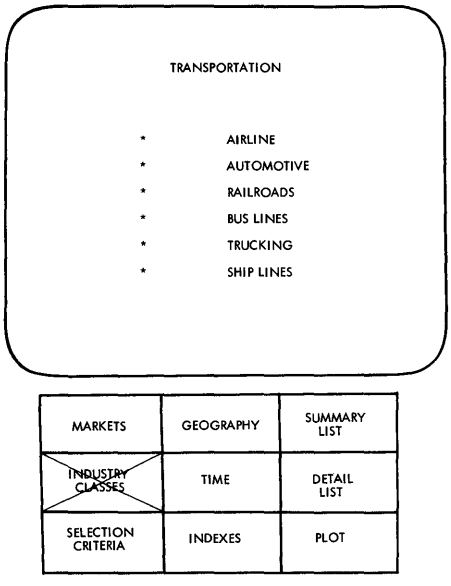


FIGURE 8
LIST DISPLAY OF TRANSPORTATION

The following statement is entered through the keyboard:

> List all automobile and airline stocks on the New York Stock Exchange which have a yield of at least 2% in 1964.

To start the process, the MARKETS key is pressed. The display shown in **Figure 6** appears; from this list display the item NEW YORK is selected. Next, the INDUSTRY CLASSES key is chosen, which results in a display of INDUSTRY CLASSES as shown in **Figure 7** . The category TRANSPORTATION is selected, which is automatically followed by a breakout of this item as shown in **Figure 8** . From the latter list, items AUTOMOBILE and AIRLINE are chosen.

When the SELECTION CRITERIA key is pressed, the format display of **Figure 9** is presented. This display permits the selection of ranges for the indicated criteria, where the left and right locations correspond to the lower and upper bounds. The number two is entered as a lower bound (leaving the upper bound blank, which by convention indicates infinity).

The TIME key is next selected, and the desired calendar period is entered. Finally, the SUMMARY LIST key is chosen and the query entry completed. The resulting output

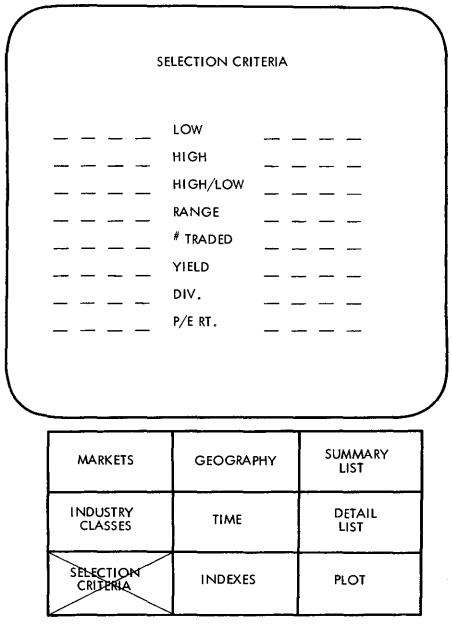


FIGURE 9
FORMAT DISPLAY OF EARNING RATIO

from this query, when retrieved, may appear as shown in **Figure 10**.

SOFTWARE REQUIREMENTS

Hardware Environment

The extent of software depends upon the hardware environment. **Figure 11** represents a very general system where the display subsystem functions are specifically identified. In particular, the display subsystem controller is shown to have the function of: message routing, data buffer, display regeneration, and character generation.

The latter two functions are generally part of each console hardware unit. Then the display regeneration is performed from a buffer memory associated with the console.

If the display subsystem controller is a computer, then it may have direct access to the data base as shown by the dotted line. The controller is then capable of performing much of the man/machine communication. In the following text it is assumed that such is true.

YIELD OF SELECTED INDUSTRIES
GREATER THAN 2

AIRLINES	
AMR	2
UAL	2.4
AUTOMOBILES	
AMO	7.1
C	2.3
F	3.7
GM	5.7

FIGURE 10

FINAL QUERY OUTPUT

Software Design Parameters

The interactions between man and machine are spread over relatively long periods of time, and are asynchronous with respect to each of the users. Hence, if reasonable response times are to be met, access must be provided to programs, pre-stored displays and data on a random basis. Because of the expected time sharing of the central processor between multi-stations for intermittent serv-

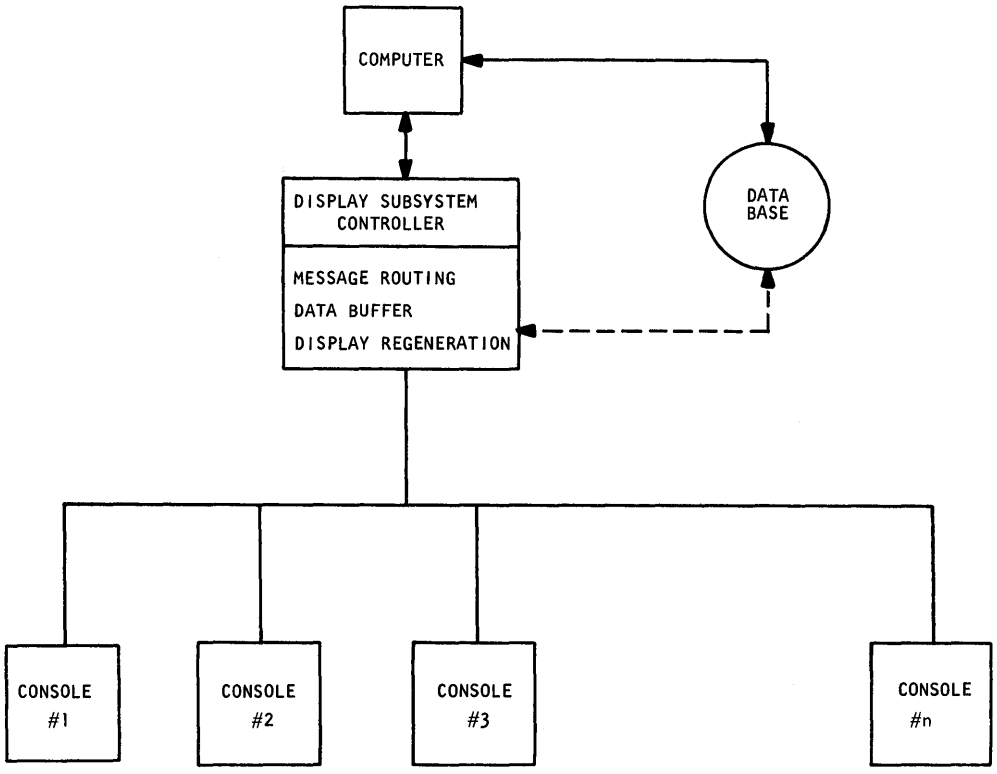


FIGURE 11

TYPICAL ON-LINE DISPLAY CONSOLE COMPUTER SYSTEM

icing, console "history tables" reflecting user transactions to the current time must be maintained. In addition, since unpredictable time lapses occur due to intermittent human responses, the "position" of a program in a particular procedure must also be maintained.

Effectively, the program must wait (or do something else) whenever a display is presented to the operator. As the operator enters data (if required), the computer must momentarily return control and monitor each entry. After the entries for a single display are completed, an appropriate "end of message" initiates the next logical step. At the end of the final step, a complete and meaningful message or direction is the basis for the computer's independent determination of what is to be done. It is thus possible to generate directions continually and to have the computer respond to them on an overlapping basis.

It is possible to separate the application oriented functions from those that are general purpose, and to apply to most on-line display system *applications* and *processes*. The division is made between the processes required in generating the message and the actual procedures for executing the action that may be called. The former concerns the mechanics of handling displays and composing messages; the latter is concerned with actual file handling, retrieval, processing, summarizing and formating. In this discussion, attention is restricted to the first aspect, the general purpose processes.

The objectives for the programming system design are four: first, to provide general capability and flexibility so that virtually all applications can be accommodated.

Second, to standardize techniques and procedures so that individual program segments or subroutines can be shared by as many functions as possible.

Third, to maintain order among contending users for the same files.

Fourth, to service each console as if its operator is the only user making demands on the processor.

Based on the above discussion, the programming system must include at least the following programs.

Display Subsystem Executive Control. This program performs the basic scanning, sequencing and queue control for servicing the on-line devices. In addition, it links to the master executive control which may be supervising the total processing.

Function Monitor. This program maintains the history tables and establishes the action sequences to be carried out as a function of the keys that are pressed.

Utility Program Package. This is a collection of service routines used primarily by the function monitor and executive control. The availability of these general purpose programs precludes the recoding of common functions.

Implementation Language. This is a language used by the application programmer in writing his program and is operated upon by the function monitor. The system must provide the programmer with the ability to express his program in both the symbolic language of the computer where each command generates one machine instruction and in higher order languages where each command generates many machine instructions. To be effective, the higher order language must be powerful enough to express the application problem in terms of the man/machine environment, usable with a modest amount of training, and readily expandable so that new commands and functions can be added.

A system such as this implies that application programmers must conform to certain coding restrictions and procedures so that all the possible programs can be accommodated. While this may seem a disadvantage, it is, in fact, helpful since it simplifies the programming (because of the existence of service routines). It also simplifies the implementation of new applications, since they must fit within the logical framework set forth by the system.

The importance of the second point cannot be over-emphasized. Without a well defined organizational and procedural philosophy, the programming design and implementation of the individual application can become a major undertaking.

Display Subsystem Executive Control

Table 6 presents the programming and storage requirements for a typical display subsystem. The real time requirements associated with on-line displays present a problem of priority interrupt handling and servicing. Hence, an executive system must be designed to be responsive to these requirements. Such

TABLE 6

PROGRAMMING REQUIREMENTS FOR A DISPLAY SUBSYSTEM

Programming Requirements	Required Storage 24 bits/word
Core Resident	
Programs	
Display Executive	250
Function Monitor	1500
Utility Programs	500
Input/Output	500
Console Programs	500
Buffers and Data	
Constants	100
CRT Image	400
Queues	150
Working Core	800
Input/Output	1600
Total	6300
Auxiliary Storage Resident Programs	
Start-up	100
Service Routines	1600
Restart	200
Total	1900
Implementer Compiler	2009
TOTAL	10,200

a program is equipment dependent in the sense that many hardware/software tradeoffs are possible.

The basic requirement of the display subsystem is control of a great number of input/outputs including; scanning the input lines for messages, refreshing the CRT's, accessing programs, displays and data from auxiliary memory, communicating with other processors that may be in the system, and maintaining timing responses for special purpose on-line character generating equipment.

Typical timing requirements range from refreshing the CRT within 20-25 ms periods, to scanning of inputs from the console keyboards every 200 ms. Unless certain hardware features are available, such as automatic interrupts and input/output buffering, the programs have to take these into account. Assuming no dependence on hardware, the executive program must maintain continuous cognizance and control over the input/output. This is done by the basic control loop shown by the dotted lines in **Figure 12**. Each of the five indicated functions could potentially generate a processing task as the cycle is traversed. For example, the tasks associated with scanning the input message lines is shown in **Figure 13**.

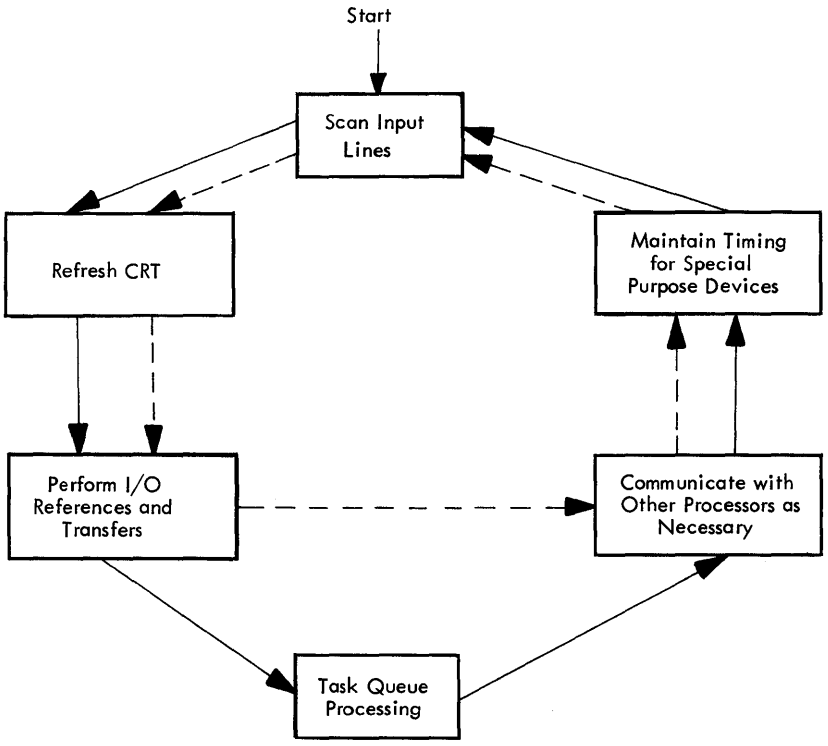


FIGURE 12

BASIC EXECUTIVE CONTROL LOOP

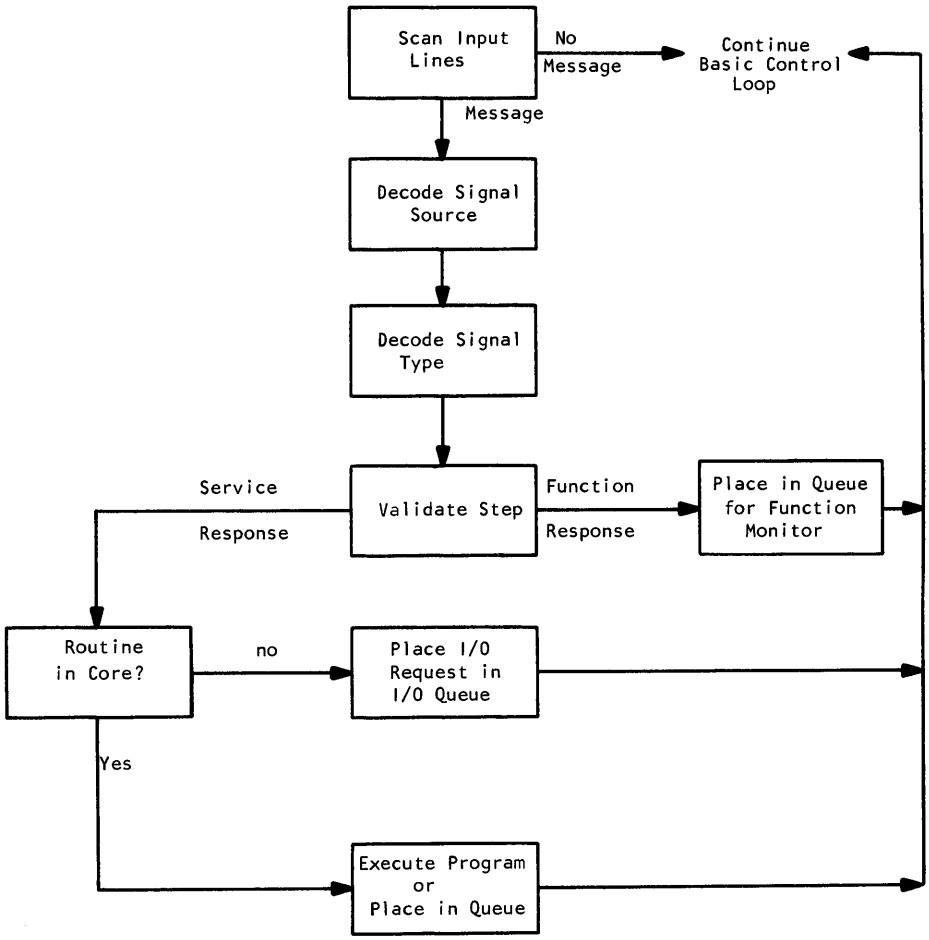


FIGURE 13

TASKS ASSOCIATED WITH SCANNING THE INPUT MESSAGE LINES

To meet real time requirements, this loop must be passed at a rate which insures return to the task which has the tightest timing constraint within a specified amount of time. This time is called the "basic cycle time." Thus, if the CRT refreshment is the critical task, then the basic control loop must return to that task within a basic cycle time.

There is also the further implication that the processing requirements for each of the five identified functions must be completed within a time which does not compromise the total cycle time.

There are three ways to do this. The first is to allow processing to proceed in increments of the basic cycle time so that temporary return to the cycle is permitted after each such segment. This leads to difficulties of recursive entries into the various processing tasks.

The second way is to spot-place a particular task in more than one position in the loop. Thus, for example, the "refresh CRT" might be placed in every other position in the loop if the other functions have a period which is very much larger than that of the CRT refresh cycle.

The third way is to permit only a minimum of processing as each of the tasks is reached, and to place in a queue those functions not completed. This queue is then processed during the residual time which is left over during every cycle. This is shown in **Figure 12** by the box which is part of the loop indicated by the heavy lines. It is, of course, necessary that the residual be non-zero enough of the time if any processing is to occur.

The following comments are presented concerning the implementation of the executive control with respect to the presence or absence of the indicated hardware features.

If neither external interrupt nor a real time clock is available, the tasks associated with

each of the control loop functions, and all other calculations, must be programmed in segments so that each segment permits return to the control loop and maintains the timing.

If a clock is available, the executive can preset it at the beginning of each cycle so that it interrupts the processing of the queue at the proper time.

If external interrupts are available, the function of the basic control loop has been absorbed by the hardware and a minimum executive function is needed. Consoles are then serviced on demand.

Function Monitor and Implementation Language

The function monitor is a specially designed program to facilitate responses to the console's variable function keys. Although not all consoles have a set of keys of this type, a general purpose console has such a set.* These keys are characterized by the fact that their labels, and also their identifying codes, can be changed by the operator to suit the application.

The process of entering information into the computer to make a request has been discussed in detail in the previous section. Different applications require different displays and different sequences of presentation, and it is advantageous to design a scheme which is not application dependent, so that the user can design his own data entry scheme and query language.

The function monitor operates on a very special implementor's language oriented to display manipulation. When one of the function keys mentioned above is depressed, the executive control recognizes this and passes control to the function monitor. It is the ease with which the implementor can specify and modify this list of statements which makes the function monitor so valuable. To illustrate the capability of the language, some of the possible statements are now given.

The first is: turn specified console lights on (off)—the lights are specified in parameter words following the instruction.

Another is: display the following characters on the CRT—the characters along with their location coordinates are listed following the instruction.

Others are: locate a display in auxiliary storage and present it on the CRT—the identification of the display follows the instruction; clear a specified buffer—the buffer area may be either pre-established or specified in the words following the instruction; and enter the specified characters in the buffer—the characters are listed following the instruction.

Finally, there is: process the "list" display —special codes (specified by the query language) are extracted from the list display as dictated by the selections of the operator and are placed in the buffer; and process the "format" display—the parameters entered by the operator are extracted from the format display and stored in the buffer.

A more sophisticated language can be designed to cover more applications. The above language, however, is indicative (along with the function monitor) of what is necessary to service the kinds of retrieval requests mentioned earlier.

Utility Programs

Utility or service programs extend the hardware so that user functions become available to the application programmer without his concern for programming them. This software is primarily concerned with making it easier for the entry of alphanumeric information on to the CRT, expeditiously. Also included are useful functions for data handling, control of displays, system control, and the detection and setting of status and error indicators.

CONCLUSION

In the past, on-line display devices were primarily of interest to the military. Now, the commercial market is becoming a serious user. Evidence of this is shown by the increasing number of demand query reservation and quotation systems and time sharing systems such as Project MAC. These applications are stimulated by the current trend to on-line and real time data processing systems.

The many possible applications of display devices suggest a need for general purpose software to assist the programmer in designing and implementing man/machine procedures.

This paper has identified the generic features common to on-line CRT displays which form the basis for software responsive to these features.

*These keys can be implemented by hardware or software. For example, if keys are unavailable or limited, they can be simulated on the CRT itself and the light pen can be used to "press" the keys.

Emil R. Borgers*

Priority Interrupt Characteristics for On-Line Control

INTRODUCTION

ON-LINE CONTROL is the basis for a broad spectrum of significant work being performed in a wide variety of applications areas. The "least difficult" applications have been attacked first—and successfully. By "least difficult" is meant those applications requiring relatively slow response times (chemical processes requiring 5-10 minute control cycles—1 or 2 human reactions—data logging). Conventional general purpose computers designed primarily for batch-processing have been the nuclei of these initial systems. The problems encountered have made the phrase "program around it" commonplace.

The "more difficult" on-line applications remain to be done. The demands implicit in the process, or simply the cost feasibility, require that on-line systems, particularly the central computer, be designed so that "programming around" a deficiency is not necessary. This is not for the convenience of the programmer since he is professionally committed to the performance of different tasks, but because the phrase implies the substitution of a sequence of program steps requiring a measurable amount of time for hardware capable of responding in an instruction time. These "more difficult" applications cannot tolerate relacement of hardware by programming because the required response times are so fast that programming implementation is not feasible.

Basically, in all on-line control systems, the central computer acts as a "transponder" (i.e., the computer performs some calculations and initiates passes on the basis of an incoming signal). Some examples of incoming signals are time sharing customer stations, clocks, emergency alarms, and management inquiries. To such incoming signals, the computer must respond rapidly and reliably. How well the computer is able to do this generally determines the maximum capability of the on-line system. A necessary ingredient in the responsive ability of the on-line system is the inclusion of a powerful and flexible priority interrupt system.

PRIORITY INTERRUPT SYSTEMS

Priority interrupt, as a concept, has been with us for a long time. Interrupt experiments were conducted on systems as early as the Whirlwind I. Subsequent command and control systems have incorporated interrupt techniques which provided the necessary capability for the not too demanding "least difficult" problems. In the last three years, however, specific interrupt techniques have been developed to satisfy the "more difficult" on-line applications. This development coincides with the appearance of production line real time computers.

To measure the effectiveness of a priority interrupt system, certain criteria must be established:

Reaction Time: this is the time between the occurrence of a signal (or request) external to the central computer and the commencement of execution of the first *useful* instruction requested by the external signal.

*Director of Programming, Scientific Data Systems

Overhead: is the difference between the total time necessary to completely process the incoming request and the execution time of all useful instructions.

Optimum Priority Response: is the ability of the central computer to correctly react to incoming priority requests. If the "most important" instruction is not being executed at a given instant of time (as determined by the priority status of the request lines), then the priority response of the central computer is said to be *sub-optimum.*

System Saturation: occurs when the on-line system cannot respond quickly enough to all of the requests. The system is underdesigned if this state exists.

TYPES OF PRIORITY INTERRUPT SYSTEMS

At least three types of priority interrupt systems are in general use today. These are the search ring method, the single-level indicator method, and the matrix control method.

Search Ring Method

The search ring method is implemented by use of an n position electronic stepping counter which continually scans n interrupt lines. The highest priority lines are scanned first so that, if two requests are received simultaneously, the higher priority line is recognized. The counter associated with the position of this interrupt is then transmitted to the central computer to be used in forming the address of the first instruction to be executed. The program counter is automatically saved and restored at the end of interrupt processing.

This method has poor reaction time since if any interrupt is now being processed, the computer is "locked out" from all other interrupts. In the worst case, the response time can be as long as the longest interrupt servicing routine in the system. Overhead, however, is not excessive, since the program counter is saved and restored. True, priority response is not present since the simple scanning technique prohibits a high priority interrupt from being recognized while one of lower priority is being processed. This method is thus said to be single level as opposed to true priority. Another disadvantage of the search ring method is its capacity. As n increases, the time to scan all lines increases. If n is large enough, either faster (and more expensive) components must be used or the scanning cycle increases, lengthening the reaction time even further.

Single Level Indicator Method

The single level indicator method is the most common (and least expensive) method in use on existing computers. Essentially all interrupt lines develop an "OR" output which serves as the interrupt request signal to the central computer. At the completion of the current instruction, the program counter is stored and an interrupt processing subroutine is entered (**See Figure 1**). This subroutine tests each interrupt line in sequence to determine which request caused the interrupt. Either by program or automatically, the interrupt line recognized is reset and program control is transferred to the correct routine.

The response time and overhead for this method are both very high since, after the interrupt request line is activated, a significant number of program steps must be exe-

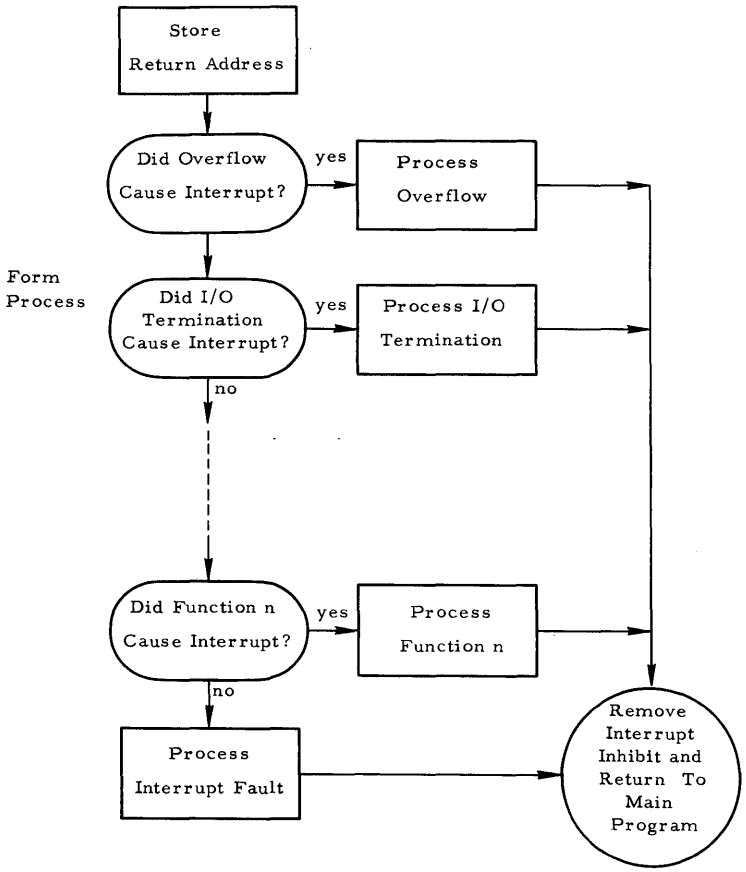


FIGURE 1

EXAMPLE OF INTERRUPT PROCESSING ROUTINE
SINGLE LEVEL INDICATOR METHOD

cuted before the central computer begins to obey the request. While the highest priority requests will be tested first, the high overhead may make the lower priority interrupts ineffective. This method is also single level since no interrupt can be recognized while one is being processed.

The combination of slow response time and high overhead *plus* the lack of true priority response makes system saturation a possibility before the capacity of the central computer has reached an economic level. Consider an m microsecond computer having two interrupts occurring every 500 m time intervals. The overhead on each interrupt processing would conservatively be 20 m. The non-useful computing time just to recognize these interrupts would take 8% of the computer's capacity. If more interrupts of lower priority are added (with their correspondingly high overhead) system saturation is very likely to occur.

Matrix Control Method

The third method, the matrix control method, provides two flip-flops for each interrupt line to be recognized. These flip-flops provide the necessary memory to determine the current status of the line. Three states are used, as shown in **Table 1** .

Table 1. States of Matrix Control Method

State	*Condition*
1	No interrupt has been requested on this line.
2	An interrupt has been requested, but has not been recognized by the computer.
3	The requested interrupt has been recognized by the computer, but has not been completed.

Each interrupt line (or level) is positioned into a matrix based on the order of priority—the highest priority being closest to the output, while the lowest priority is the farthest away (**See Figure 2**). An interrupt request being received at a given level automatically causes the level to shift from state 1 to state 2. If no higher priority level is present in states 2 or 3, the matrix permits the interrupt

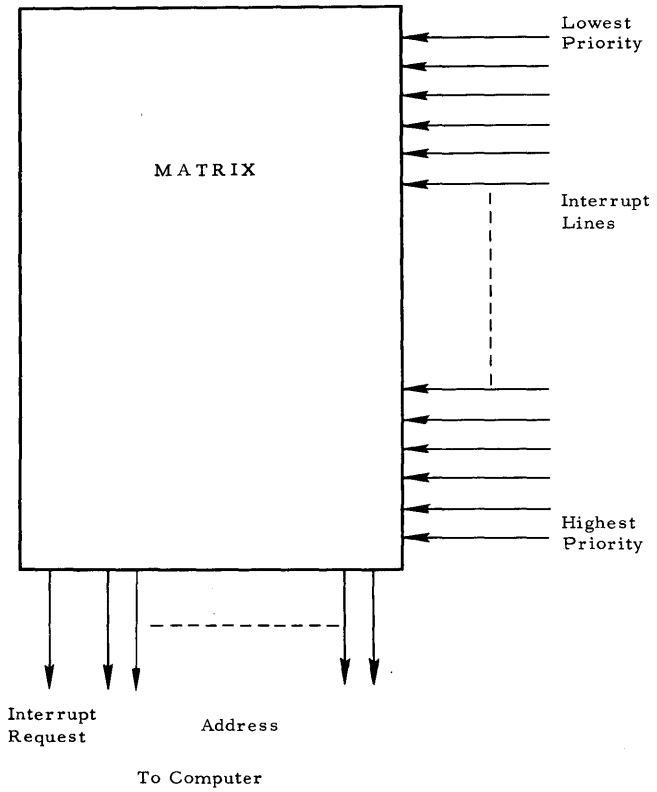


FIGURE 2

MATRIX CONTROL METHOD

request line to be activated to the central computer. At the same time, an address unique to the requesting level (determined by diode selection) is supplied to the computer. At the completion of the present instruction, the computer transfers control to the memory location determined by the provided address. At this point the program counter is preserved and a signal is sent to the priority interrupt system to change the state of the highest priority level in state 2 to state 3 (by design, the requesting interrupt level). At the completion of the desired routine, a unique instruction returns control to the point of departure, simultaneously signaling the priority interrupt system to change the highest priority level, at present in state 3 to state 1.

The matrix control method provides both a short reaction time and low overhead. Every interrupt request is obeyed immediately provided no higher priority request is in execution. A favorably low overhead is achieved since no program time is used to transfer to and return from the desired routine. The biggest advantage of this method, however, is a near optimum priority response. The most important instruction is being, or is about to be, executed at any instant of time. When the required routine for any interrupt level is completed, the matrix control method ensures that the next routine to be executed will have the highest existing priority regardless of how many lower priority requests remain only partially completed.

The effect of the matrix control method may be achieved without the structure described here. The required elements are a) memory for each interrupt level, b) a hardware priority structure, and c) central computer communication to inform the priority interrupt system of a change in state. To conserve cost, interrupt levels have been "shared" by a number of interrupt lines in some systems. Since the lines must not interact, the on-line systems designer must take this approach with caution.

CURRENT-STATUS PRESERVATION vs. OVERHEAD

So far, the problem of preserving machine registers (other than the program counter) has not been mentioned. This problem is discrete and exists regardless of the interrupt recognition methods mentioned previously. If the routine executed in response to an interrupt request uses (or destroys) any of the machine registers (such as the accumulator, index registers, overflow indicator, etc.), the contents prior to use must be preserved and restored after completion. The problem is likely to become more complicated since computers are being designed with more and more machine registers for greater flexibility. Approaches used so far have been: a) let the program decide what to save and restore, b) implement through hardware an automatic store sequence to save registers in memory and automatically restore after completion, and c) maintain all registers in memory and provide multiple register groups for each interrupt routine and the main program (when an interrupt occurs, a pointer automatically selects the unique register group). The first approach is generally effective if the instruction set is designed to allow many operations to occur without affecting any (or few) registers. In this way, the programmer has alternative choices to keep the overhead low. The second approach involves a fixed overhead no matter what functions are performed in the interrupt routine. The third approach gives a very desirable flexibility to the interrupt capability. If, however, the registers are accessible at the same cycle time as instructions, the tendency is to slow the whole computer down just for the sake of interrupt capability.

Future computer design, representing a departure from the conventional structure, must effectively solve this problem and reduce the overhead even further than is now done. While this factor does not reduce the efficiency of the computer as a transponder as much as some of the other factors, it limits the maximum efficiency obtainable in on-line systems. Ideally, the overhead time should approach zero.

INTERRUPT ARM/DISARM CAPABILITY

In more complex on-line applications, the requirement occasionally exists for inhibiting recognition of some interrupt requests while other functions are being performed. For example, a high speed transmission such as a disk transfer, which may have low priority until the instant that the request has been initiated, must capitalize most of the computer time. If the priority structure is allowed to stand, higher priority items might inter-

fere with the transfer and cause transmission errors due to loss of information. On the other hand, the system may require that certain critical interrupt lines remain open at all times so that the prevention of all interrupts is not feasible. This situation requires the use of an interrupt arm/disarm capability. To do this, a flip-flop is placed on each required interrupt line external to the interrupt control system. Each flip-flop must be under control of the central computer. When the flip-flop is SET by the central computer, interrupt requests can be recognized and the interrupt line is said to be armed. Correspondingly, when RESET, interrupt requests are inhibited and the interrupt line is disarmed. To conserve computer time, interrupt arm/disarm flip-flops are usually placed in the desired state in multiple groups rather than singly.

PRIORITY INTERACTIONS

Assignment of priority interrupt levels to particular functions in a given on-line system is, at times, an interesting and perplexing problem. At first, it appears that the systems designer should order the request functions on the basis of importance and assign levels accordingly. This, however, produces the most effective system performance only by accident. Priorities must be assigned using the interaction of functions with each other as a primary basis.

Consider a simple on-line control system with three major requirements: (1) receive and modify input data, (2) output these data, and (3) maintain time in milliseconds. The estimated length of execution of these functions and the worst case frequency of occurrence is shown in **Figure 3**.

In this hypothetical case, the maintenance of time is for future off-line processing and is the least important of the three functions. Since the output of data is possible only after data input and modification has occurred, system input is the most important function. If priority were assigned strictly on the basis of importance, the time function will be missed under certain conditions. This erroneous time measurement cannot be tolerated. If, however, the time function is assigned the highest priority (as shown), no time information will be lost. More important, the net effect of this assignment is to prolong either system input or system output by a few microseconds. Since worst case conditions are shown, no serious problem results and system saturation is avoided.

The on-line systems designer must ensure that all possible interrupts in the system are operating compatibly with each other when worst-case conditions occur. Debugging on-line systems with incompatible priority interrupt assignments is, at best, a horrendous task. These problems must be solved during

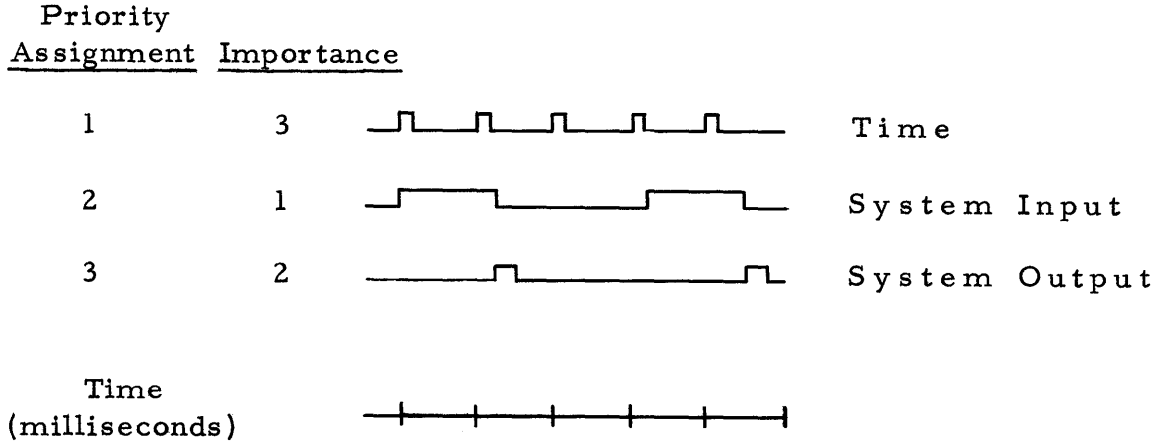


FIGURE 3

EXAMPLE OF PRIORITY INTERACTION

the design of the system—not during program and hardware checkout. System interactions involving priority interrupts are often not observable during system checkout.

Even when trouble is detectable, it presents to the human a behavior pattern similar to an intermittent component failure. This generally leads to many false and frustrating excursions before a solution is found.

DYNAMIC PRIORITY REALLOCATION

Based on the occurrence of certain events, it may be necessary to reassign the priority levels of key interrupts dynamically under program control. This requirement is fairly common in military command and control systems resulting from a change in the tactical situation. This capability has been implemented in a number of ways on different systems. Implementation has ranged from large banks of flip-flops to core switching matrices. Typically, a large amount of expensive hardware is necessary if total flexibility is required.

If only a few different options of dynamic priority reallocation are required in an on-line system, it is usually cheaper to assign a given interrupt request line to two or three different priority levels in the interrupt system. Coupled with arm/disarm capability on each line, each request can be reassigned to a different priority level as the situation changes. This method is generally less expensive than the more flexible approaches.

FUTURE REQUIREMENTS

At least two advances are required in the priority interrupt area to make effective use of the higher performance hardware being developed for on-line systems use. They are time related priority assignments and externally weighted priority.

Time Related Priority Assignments

Existing techniques are not adequate to handle a phenomenon which is appearing more frequently as on-line systems become more complex—the time related priority assignment. Consider an on-line system with a low priority task requirement of once each second. Immediately after this function has been completed, it should have the lowest priority in the system. On the other hand, if a whole second has passed since execution its priority should be high. What is required then is a technique which allows this function to "creep" upwards in priority as a function of time. This case continually exists in time sharing systems as a given user would like to "wait in line" for his next turn even though some users have a higher overall priority. Present hardware to implement this requirement is very complex and expensive. A software approach, while technically feasible using push-down lists, increases overhead prohibitively and reduces overall efficiency.

Externally-Weighted Priority

In on-line multiprocessing environments, separate tasks are being performed in a number of computers, the results of each having only a partial effect on the entire system. Priority requests between computers are more effective if the requesting computer can "qualify" its importance based on the situation. The receiving computer can then use this qualification as a weighting factor to determine the ultimate priority of the request.

Certain time sharing systems involving many users will require a similar capability. Let us give the user a number of levels of service, selectable at his console, for which he will pay different usage rates. The different service levels would provide the user with different priority usage of the time shared computer, either more frequent access or longer on-line time at each access.

To provide efficient on-line systems of the type provided above, externally weighted priority must be available without increasing overhead.

CONCLUSION

The design of an effective priority interrupt capability has not always been proportional to its contribution to the overall on-line system. A weak priority interrupt (or none at all) can reduce the number of useful instructions executed by the central computer to as little as one-half the total. Excessive computer power must then be employed to compensate for the loss of computability. As on-line requirements become more complex, an increased burden must be assumed by use of priority interrupts. Adequate solutions have not been found to meet all foreseen requirements. Yet, we are learning to make effective use of what is available to make the digital computer properly react to the on-line environment.

Arthur M. Rosenberg*

Group Communications In On-Line Systems

INTRODUCTION

THE USE of a digital computer as a personal, on-line tool is being successfully achieved on a practical basis through the use of various time sharing techniques, and another door has been opened to new and more interesting applications of our computing technology. To achieve this, time sharing brings ready accessibility and suitable response time to bear upon the human user's requirements for direct employment of computer services. Thus, personalized conversation or "on-line" interaction between man and the power of a computing machine offers fruitful solutions for today's complex and dynamic problems.

Much has been said about the many advantages to be gained by allowing human users to have direct contact with a computer. This is primarily true because most computer users previously were required to operate in a job-shop environment, very often through one or more middlemen, programmers, computer operators, keypunchers, etc. This may have proved a frustrating and discouraging experience to the non-programmer, and, in many cases, even to the programmers. The taste of freedom and power felt by the on-line user of a large computer system can now be realized by many rather than a limited few, and this sensation has caused many devotees to hail the dialogue of man and machine.

Communications have been, and will continue to be, closely allied to the fruitful use of computers. In time sharing systems, remote computing and multi-processing computer networks depend heavily on various communications facilities. By the same token, however, sophisticated communications service will also be provided by the computers in new electronic switching centers, and time sharing systems have a vested interest in some of these services. Personalized or "on-line" computing is a very powerful service; it becomes even more powerful if it serves a group of individuals involved with a common problem. This is, fortunately, not only achievable within the framework of time sharing systems, but may be done with relative ease. A time sharing system, with its communication lines extended to a number of user consoles, is, in effect, a communications switching center not only between the consoles (or other computers) but also for user programs. This last feature adds the facility of interactivity in the time sharing environment.

APPLICATIONS OF GROUP COMMUNICATION TECHNIQUES

Man is a gregarious animal, and a machine, albeit a computing machine, is still, after all, a machine. The social environment of problem solving has often emphasized the concept of two heads being better than one. Now that a personalized computer is available, which may act as a helpful and efficient assistant, the popular phrase might be changed to: "Two heads and a computer are better than one head and a computer."

There are a number of computer applications in which two or more people will inter-

*Member of the Technical Staff, Scientific Data Systems.

act in some way with each other as well as with a computer program or programs. Variations may occur in the manner by which communications between participants will be controlled and these will be discussed later in this paper. However, at this point, let us briefly review some of these applications.

Experimental Games

Time sharing communication techniques are essential to computer based experimental games with human subjects. Programs for such experiments must be designed for multiple users, must control and restrict station input and outputs, and must allow the experimenter or umpire to time the experiment, observe all participants' input actions, and intervene when necessary. At his station, the experimenter can specify which stations participate in the game, and may alone use the full input and output capabilities of the experimental program. Also, he alone may issue executive commands to the time sharing system during the course of the experiment.

Group Debugging or System Testing

Since computer programs are growing in size and complexity, it is usually necessary to partition the development of a large "system" among several programmers. Eventually, the program components will require checkout as a system, and the programmers must debug their code together. As the group watches the operation of the program at individual (and remote) system consoles, and a fault is detected in an area of code, the person responsible for the particular code can use on-line debugging facilities to fix the trouble; if the problem is more complicated, the group members can then discuss the situation directly in a "conversation mode" between stations. As a last resort, the group can call in an "expert," as described below.

Consultation

Many types of programming services will be available to users of the time sharing system; some fairly complex to a neophyte. The time sharing system executive can respond to a call for "help" in particular types of operations, providing an expert "teacher" for language systems, the time sharing system itself, mathematical and statistical routines, etc. First, the expert may communicate directly with a user via a "conversation mode" to discuss the problem; then he may find it practical to have the user re-enact the operation that caused difficulties. The user's operation can be put into a "trap" mode by the system, allowing the expert to correct the user's inputs before processing takes place. Alternatively, the expert can demonstrate an operation for the user, with the latter playing a passive, observer's role. Whichever the alternative, the stations of the user and the expert would be linked by the time sharing system.

Demonstrations

The on-line technique of consultation described above is also effective for educational demonstrations to groups of users. Here the demonstrator links his station channel to the users' channels so that they can all see his inputs and outputs. He may accompany his operation by narration through a "conversation mode" facility, which encompasses all the observing stations (party line). The observers may at any time send questions to the demonstrator's station as well as to all the other observers. Further, during or after the initial demonstration, the observers may use the consultation mode to perform the operation themselves, under the guidance of the expert.

Briefings

Communication facilities also serve for conferences on operational information in a user's system. One or more individuals control such briefings, calling upon the operational object program to present an information display to the various stations. Normally, only the briefing control station can have inputs to the operational (briefing) program, but all stations will receive display outputs. Briefing control, however, can be dynamically delegated to any appropriate station in the group, when necessary.

Since much of a briefing is often "canned" in advance, users may prepare their briefing displays during a dry run, attaching retrieval "labels" to specific data presentation packages, and, if necessary, calling for a basic sequence of display of such packages. The sequence does not have to be strictly followed, since the user can always call directly for any labeled display, or dynamically create a new, unlabeled display.

Common Data-Base Maintenance and Retrieval

Common to almost all large information systems are the acquisition and retrieval of

data from many sources and users. Operational communication is effectively performed indirectly through a common operational data base, the object program system servicing a group of users by either updating the data base or by retrieving information from the data base. Here, a non-replicating program design is advisable to service many users in a parallel fashion without costly program redundancy. Operational controls over the program system can limit input of new data to some stations, other stations can retrieve data, and some stations can do both. Security and quality controls are thus implementable via the program system.

Technical and Administrative Management of a Computer System

In a large programming installation, many computer users are engaged in developmental and production activities. An outstanding problem is often the technical supervisor's difficulty in maintaining current awareness of what his programmers are doing, such information being necessary for scheduling, priority assignment, technical evaluation and guidance, etc. The time sharing system, while serving its full complement of users, can also permit a supervisor station to monitor individual stations and provide on-line managerial guidance. The accounting function of the time sharing system can provide historical data on each user, valuable to the technical supervisor for assigning priorities to his programmers and directing their on-going activity.

Computer Operator Activities

One of the more mundane communication requirements rests with the need for computer operator services. Users of a time sharing system must be able to communicate with the computer operators for tape handling functions, disposition of off-line outputs, machine problems, system status information, or special communication arrangements. Likewise, computer operators must be able to communicate with users to notify them of system conditions affecting their operations. Although much of this communication can be carried out indirectly via the time sharing system executive and object programs, unanticipated problems require direct communication with the computer operator.

In many ways, the human elements in the man-machine relationship derive significant benefits from solving problems and performing operational tasks as a group. The software and hardware facilities of a time sharing system must therefore enable the users, including the computer operator, to communicate and interact freely, under a variety of selective controls, with each other and with the computer programs.

ELEMENTS OF USER COMPUTER COMMUNICATION

Many types of communications can be found within the confines of a computer system and this is particularly true for time sharing systems. We might think of users (people), programs, computers, special devices, and systems talking to other users (people), programs, etc., etc. However, if the proper foundation is laid for communication between these elements, there will be no limit to which the thread of interaction may be extended.

Communication between a computer and a human being is not as flexible as between two humans, but improvements are rapidly being made in this direction (problem oriented languages, etc.). The computer can now be made more welcome as a powerful resource and introduced into the problem solving tasks of the human world. The problem, then, is to clarify and extend communication linkages between the human group or organization, and the computer programs in the on-line, time shared environment.

Man-Machine and Man-Man Interaction

The most basic types of on-line interaction and communication in the time sharing system are those in which the humans directly interchange information between themselves and those where the interaction takes place between "man and machine." When we speak of MAN-MACHINE relationships with a digital computer system, we, of course, really mean a MAN-PROGRAM dialogue. Let us examine the mechanisms available to the time sharing system users for both of these basic forms of interaction.

Figure 1 illustrates a user station, usually remotely situated, equipped with some of the current on-line communications hardware. All the devices shown are not necessarily required, but one can conceive of effective applications for all of them.

A single channel (or unique set of channels) between the man and the program per-

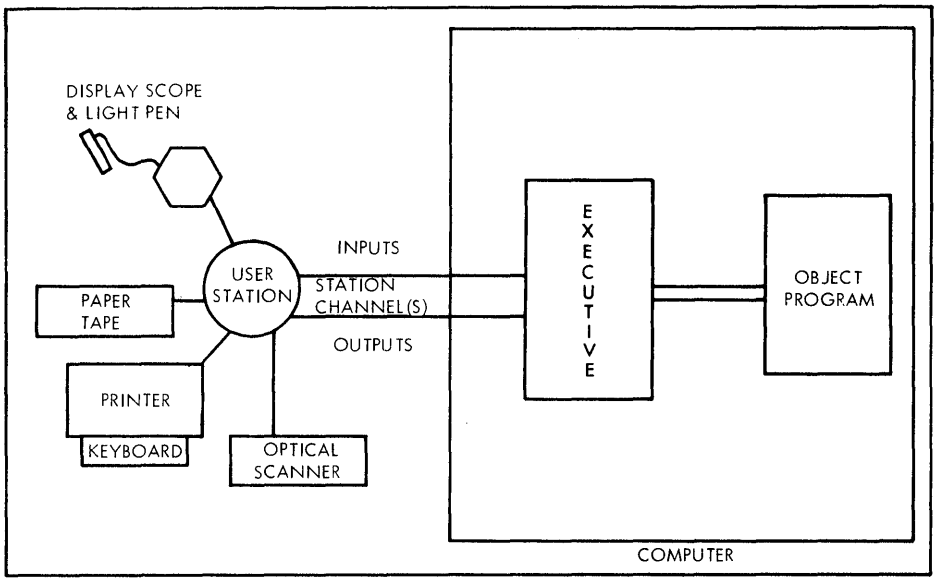


FIGURE 1

USER STATION COMPLEX IN THE MAN-MACHINE RELATIONSHIP

forms the basic man-machine operation, all inputs and outputs from various input/output devices being confined to an individual user and not normally accessible to other users in the time sharing system. **Figure 2** shows a system with a number of users.

Such arrangements are adequate for the "lone wolf" researcher or programmer, since maximum privacy is provided for the intimacy between a man and his program. However, in order to breach the communication gaps between user groups, the system executive stands out rather logically as being in a strategic action position.

We cannot assume that in a time sharing system environment the conferees are normally sitting side-by-side communicating directly. Thus, MAN-MAN conferencing will proceed either by some communication facility independent of the time sharing system (e.g., a telephone network, intercom, etc.) or through the system itself (**Figure 3**).

MAN-MAN communication provided by the time sharing system itself is by far the most interesting and powerful approach, for we can then integrate group activities with the many applications of a digital computer. The computer becomes a communication center in

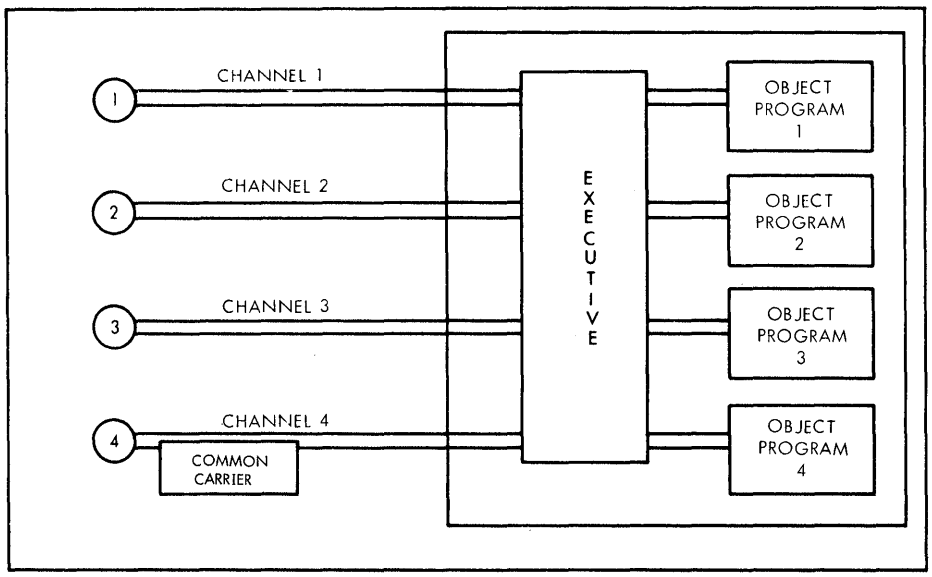


FIGURE 2

MULTI-USER SYSTEM

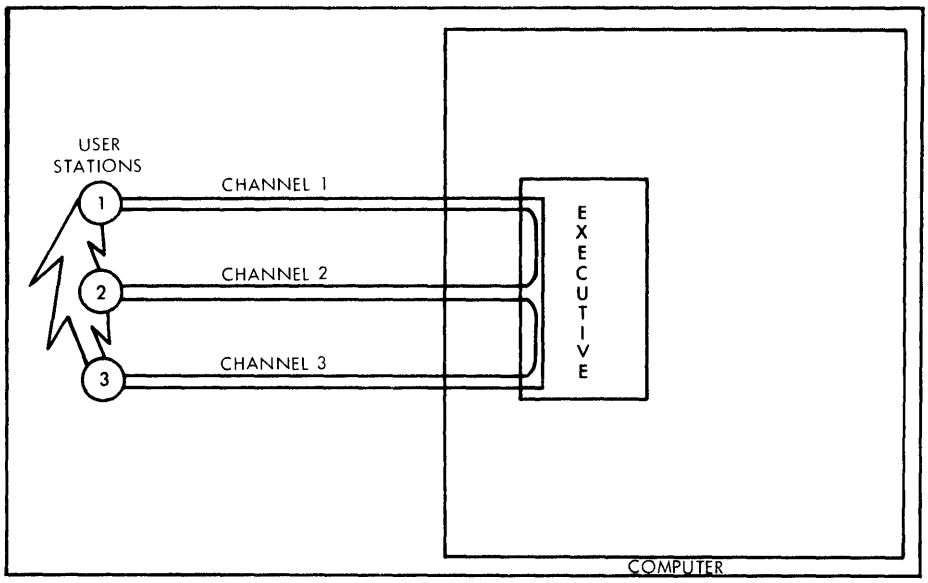


FIGURE 3

MAN-MAN COMMUNICATION VIA TIME-SHARING SYSTEM OR EXTERNAL FACILITY

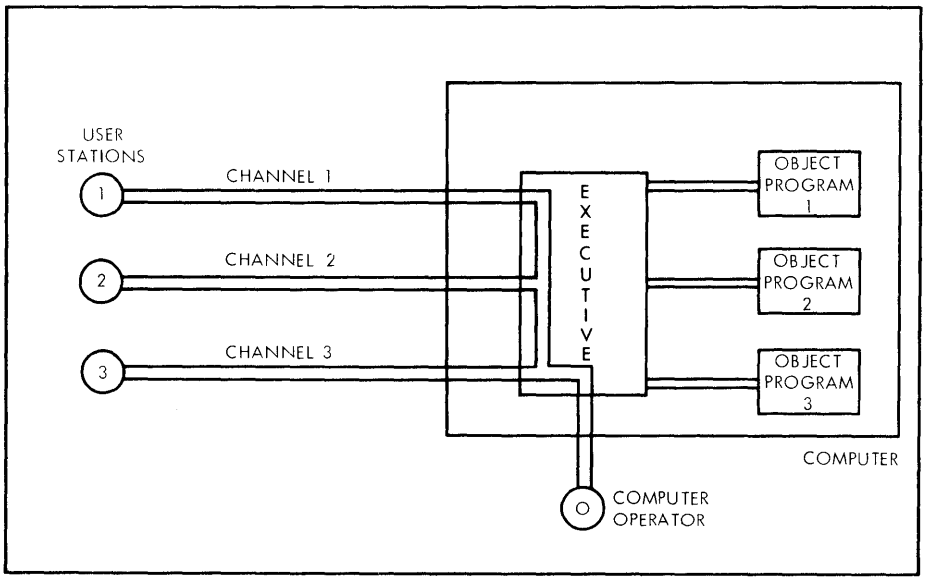


FIGURE 4

MAN-MAN AND MAN-MACHINE COMMUNICATION LINKS IN A TIME-SHARING SYSTEM

addition to performing operational data processing functions. Assuming that all MAN-MAN conferencing will normally relate to the users' data processing tasks, it is desirable that a generalized time sharing system should provide selective communication service functions in response to various user operating needs.

In general, users of an on-line, time sharing system will expect the capability for both the basic MAN-MACHINE *and* MAN-MAN communications, as shown in **Figure 4** . If the latter type of communication is not provided, the system will be operationally constrained and will lack a significant degree of flexibility.

Figure 4 shows the role of the time sharing system executive in providing overall communications service to people and programs. Using the system, a user can communicate not only with his object program (and the

system executive) but with any other user station, particularly the computer operator.*

The communication flexibility thus available will help resolve those many unanticipated problems which always manage to show up in the best of computer systems.

Although **Figure 4** implies a single processor to perform an operation in a time sharing system, in reality a multi-processor capability is desirable for other than very small systems. In particular, many of the communication functions discussed in this paper may be profitably delegated to a separate processor which is allied with the system executive.

Men-Machine Interaction

Going beyond the basic MAN-MACHINE relationship described above, the time sharing system must permit object program communication with more than one user station channel, in order for it to interact with a group of users. Such group interaction with an object program in a time sharing system is one of two types; the first type involves object programs designed for many users, the second concerns normal, single user programs. Responsibility for the first type of group communications will rest primarily with the object program, not with the time sharing system executive. For the second type, the executive of the time sharing system can facilitate the additional group communications.

Type 1. *Multi-User Object Program*

Two examples of multi-user programs are: experimental games with human subjects, and public service programs. For experimental games, the programs are used by a designated set of individuals; public service programs, on the other hand, are used by a randomly varying number of users in a time sharing system. The game playing type of program, via the time sharing system executive, must sequence the input-output activity of particular user stations; the public service function will usually accept any user station randomly*. Both types of programs, however, will be required to maintain proper storage separation of inputs and outputs and contextual data for each user station involved. Furthermore, for both types of operations, the time sharing executive must permit these programs to be linked to more than one user-communication channel (**See Figure 5**).

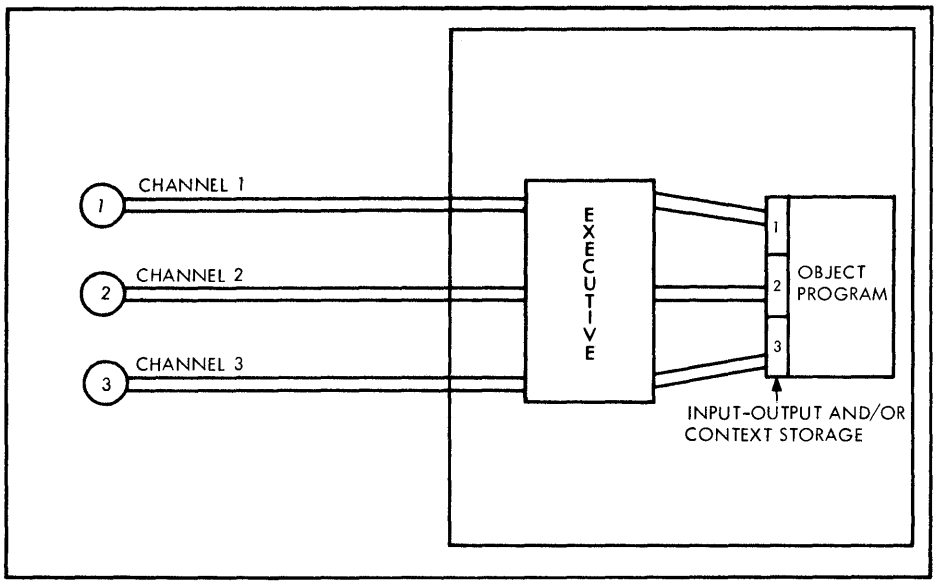


FIGURE 5

MULTI-USER OBJECT PROGRAM

* Hard copy communications are best suited for the computer operator servicing many users, since he can answer queued requests as time permits. Voice communication requires time for listening and making notes as well as limiting the operator's attention to one user at a time.

* Exceptions to this rule may be certain production functions (compilations), run only by the computer operator's station. In such cases, only one such job may actually be serviced at a time.

The terms "pure program", re-entrant or transparent routine have been used to describe the design of a program that can accommodate a number of independent users simultaneously. (The general service program is an example.)

The pure program does not modify its own instruction code, it maintains all contextual information in reference storage, and the time sharing executive provides the proper environment for each user channel to the object program at operation time. This technique permits the program to be interrupted at any time and to process a number of user requests nonsequentially. The pure program approach, although primarily useful for independent users' interaction with a service type program, can also be used for common group operational functions, such as staff activities involving on-line data acquisition and retrieval. Time and storage demands on the system will thus be reduced by the common use of a single processor rather than by replication of routines for each user.

Type 2. *Programs Designed for One User Operation*

Many object programs are designed for an individual user; if more than one user requires the use of such a program, it will simply be replicated in the time sharing system. There are, however, numerous instances when such a program should be accessible to more than one user channel simultaneously *without* replication. For example, when the program is complex enough to require checkout analysis and debugging by a group of responsible persons (system test), or when operational control of a program system must reside with a select group of limited size. In effect, what is needed is a "party line" communication link to the program.

For a one-user program to operate with a group of users, we must "patch" the station channels of the participants. The user group thus can confer with the object program, and, depending upon the selectivity of the patching, all individuals in the group can monitor each other's inputs and all program outputs.

The executive-performed channel patching shown in **Figure 6** has distinct advantages: it does not require manual intervention; it does give selectivity to the user, i.e. the user's channel can be set to receive output copies only, receive copies of other inputs as well, insert inputs, or exercise master control over other channels.

Alternative to executive program patching is the "looping" of user channels on the

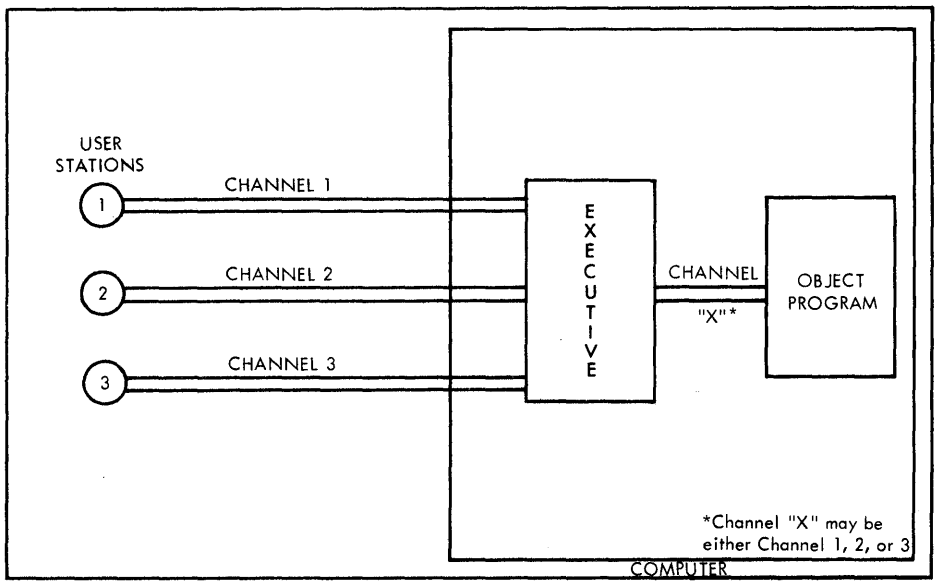


FIGURE 6

SINGLE-USER OBJECT PROGRAM PATCHED FOR SEVERAL USERS

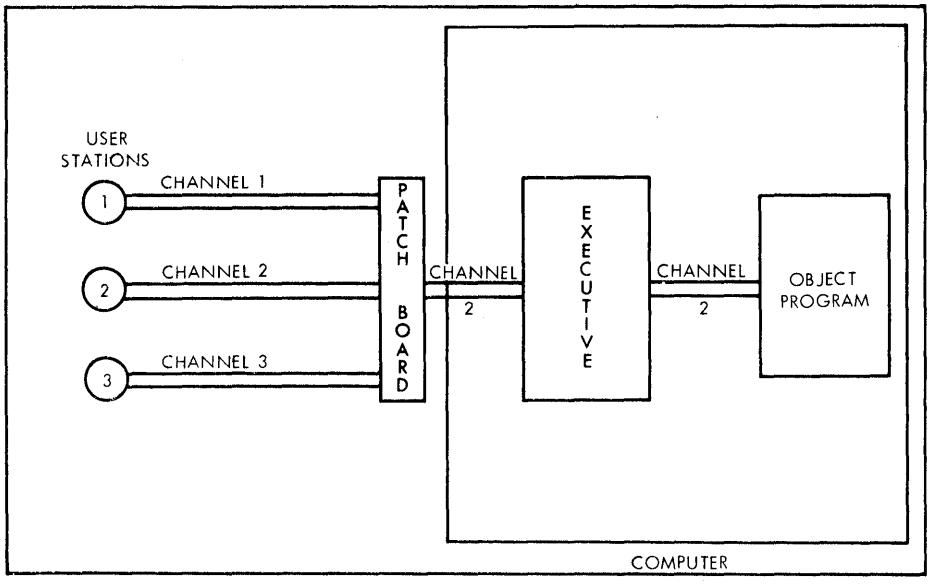


FIGURE 7

"LOOPED" STATION CHANNELS FOR SINGLE-USER PROGRAM

equipment side which provides rather limited, unsophisticated capabilities. (**See Figure 7**.)

A critical requirement for the Type 2 situation is the provision of proper operating procedures. In short, the users must know how to interact in an orderly manner with their program and between themselves, since neither the object program nor (to a large extent) the executive can really control the sequence of multi-station input and outputs. For example, organized procedures might invoke a user round-robin order of inputs controlled by the executive, or direct, MAN-MAN party line communication can be employed prior to any computer inputs. This is an interesting problem area which requires much "idiot proofing" in its solution.

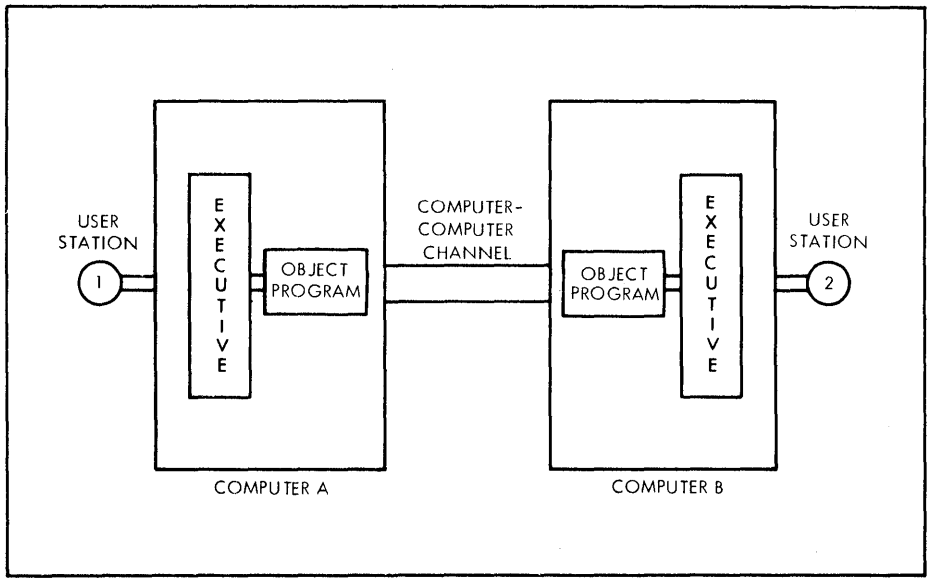
Man-Machine-Machine-Man Interaction

Just as two individuals interact directly in a problem solving task, two or more individuals with "computer assistants" can also interact profitably. Here, the human communication can be direct, as described earlier, except that two or more programs provide the communication linkage and may act as intermediate processors of the data being referenced by the users.

Figure 8 shows such an arrangement, where the communicants and their programs

FIGURE 8

MAN-MACHINE-MAN COMMUNICATION



are not resident in the same machine or necessarily in the same system.

The object programs may converse with each other via the good services of the all knowing executive, in much the same manner as input/output service is provided.

Computer-to-computer communication avails the remote user of the full input/output capabilities of *his* machine, rather than restricting him to the particular type of processing or input/output capabilities of the remote computer (e.g. the Teletype). This capability is particularly necessary for dealing with large amounts of data than can, for example, be scanned on a display scope. If the computers involved are similar, binary programs and corrections can even be interchanged from one to the other.

IMPLICATIONS FOR TIME SHARING SYSTEM DESIGN

To achieve the system communication capabilities described, we must recognize implications for the design logic of a time sharing system. The actual design techniques will vary with the type of computer and communication devices being employed and with operational programming constraints.

Communication Modes

In communicating with the computer, the user will address a program level or other users. By program level we mean the programs or parts of a program with which we desire to communicate. In a general time sharing system we normally have: executive programs—basic control or service functions, and object program or systems—operational functions.

As shown in **Figure 9**, we may find levels on levels of input communications.

A simple two-way switch for communication modes may be provided, allowing inputs to either the executive programs or object programs. A third mode, immediate MAN-MAN communication, provides a useful extension, for with this mechanism a user can simply indicate the addressee station once. Now, whenever he wishes direct communication with these stations, he inputs the "conference" indicator and his message goes through the system directly to the previously referenced stations. This mode switch will facilitate rapid shifting from MAN-MACHINE to MAN-MAN communication and interaction.

Message Queuing, Control, and Addressing

As in any communication network, a time sharing system must control the acceptance and routing of MAN-MAN conversation messages, particularly in respect to the competition between such messages and MAN-MACHINE (program) output communications. The system executive must store MAN-MAN messages until the receiver's channel is free. Depending upon the size of the buffer storage and the traffic volume, a sender may get a "busy" signal because buffer storage is unavailable. Further, it is desirable to allow the receiver to schedule the delivery of such a

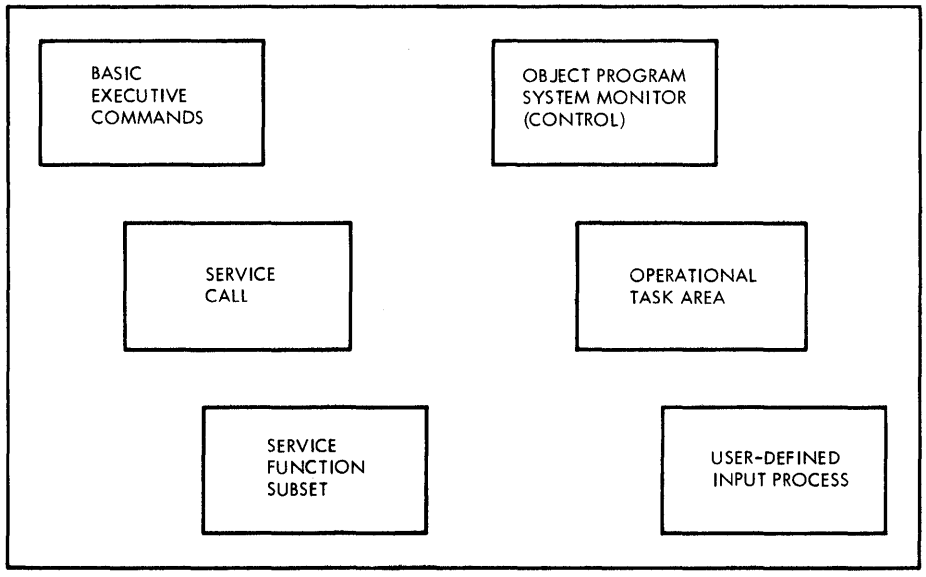


FIGURE 9

PROGRAM COMMUNICATION LEVELS

message. Thus, he might tell the system that he is "busy," and no extraneous messages would be delivered to him directly. He would, instead, be alerted if a message were waiting, and could subsequently allow transmission at his convenience.

A user's communication "address" is usually the physical channel connecting his station to the time sharing system. Even though several users share a station in the time sharing system, hard copy messages can be delivered to a station for further hand delivery to the specific individual concerned.

An established time sharing station network can have mnemonics for "administrative" and "technical" station addresses: "dial" O for the computer operator, S for special priority scheduling, C for complaint department, etc. A station directory would be necessary in the system to reflect dynamic updating of station addresses as organizational responsibilities are shifted.

Multiple addresses, of course, should be an available feature for group communication. A message can thus contain several station numbers and all would receive copies of the message. For the convenience of the system supervisor or the computer operators, an "ALL" address permits sending broadcast notices to all users in the system.

Scheduling Considerations

Time sharing systems operate on the principle of allocating computer time to various users on some equitable basis. In the basic MAN-MACHINE relationship, a given program would be operated for a single user. This allows a one-to-one correspondence between allocated time for a program and its user. However, when a group of users is associated with a single program, the question arises as to how much operating time should be given to that program. If time allocation is really determined on a *user* basis, then a program should be allowed to operate for n times the normal "quantum" (slice of time allocated per queue cycle), where n is the number of user stations linked to the specific program.

This can easily be done if the scheduling queue is station oriented and the object program is linked to each of the associated station channels. (**Figure 10** .) If a simple patching is performed, where the stations are really linked to *one* channel (**Figure 11**), the situation is more questionable for providing additional operating time to the program. Essentially, operating quanta are allocated on the basis of an on-line response cycle (e.g. maximum 2 seconds), wherein each program should service its user at least once. If a program provides "simultaneous" service, as in Figure 10, n quanta should be made available to the program. In the case of **Figure 11** , *sequential* service is being offered to n users. In the latter situation, only one quantum is necessary per response cycle, since only one of the n users can interact properly each time.

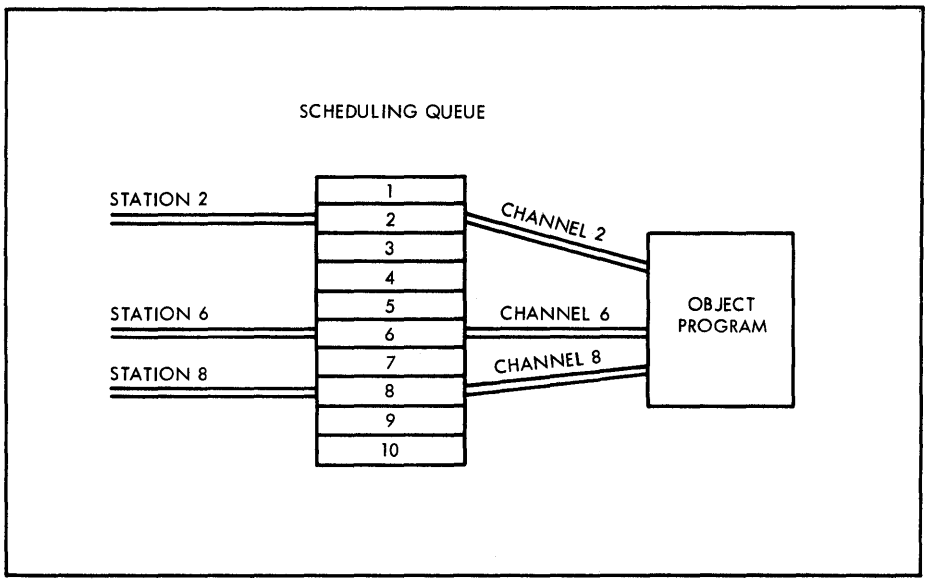


FIGURE 10

MULTI-USER PROGRAM SCHEDULING

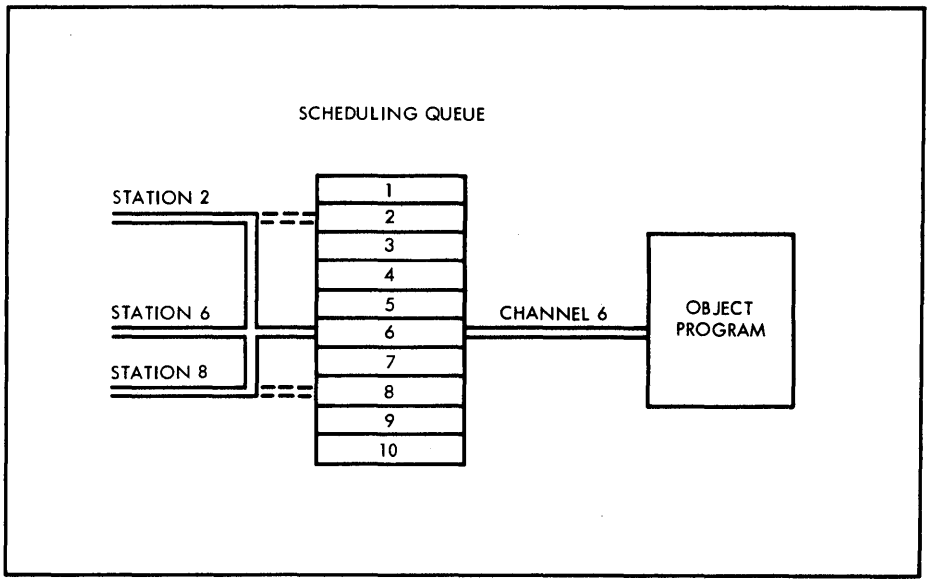


FIGURE 11

SINGLE USER PROGRAM SCHEDULING

CONTROL OF GROUP COMMUNICATION AND INTERACTION

Assuming that MAN-MACHINE, MEN-MACHINE, and MAN-MAN communications should all be provided, we must also admit that it is necessary to control such communications in operational group activity. Since the computer system acts as a central switching center, any controls and restrictions over the group operation can be supported or enforced by the time sharing system executive in conjunction with the object program.

We must consider the following control questions: Who can originate the activity? Does the originator designate the participants? Will any participants be restricted to specific input or output media? Can the originator (control station) dynamically change the operting restrictions on individual participants? Who may terminate the entire operation? Can an individual station terminate its own participation without disturbing the group? Can any particular station in the group receive priority processing of its input?

We can properly control a group computer operation only when the executive program links the user channels. If the participating stations are looped by external hardware methods, such that the object program deals with only "one" physical station channel, then no selective control can be performed, except possibly by very awkward means. For effective control functions in a group operation, the object program must be designed for multiple user channels. The station originating the activity can control the object program functions in terms of initiating and terminating its operation in the time sharing system and specifying participating stations with associated input or output restrictions. Other types of control can then be made possible primarily by appropriate design of the object program and executive communication services.

EXPERIENCE WITH GROUP INTERCOMMUNICATION

There are not many large, generalized time sharing systems in existence, and specialized multi-station systems (e.g. reservation systems) are usually confined to either single-channel inquiry or data acquisition operations. Therefore, much of the full potential of group on-line interaction with computer programs remains to be explored. However, some logical aspects of group communication facilities have manifested themselves, particularly in TSS, the time sharing system created on the AN/FSQ-32 computer by System Development Corporation. This system has both local teletype stations and terminals for remote users employing common carrier facilities. In ad-

dition, a 2KC terminal is available for remote computer utilization.

The three basic types of inter-station communication provided in TSS are: JOIN, DIAL, and LINK.

JOIN

Several stations may be joined to a common program only through the program itself. Primarily a service for experimental games, joining may be initiated by the originating station (through the game program) to include other specified stations. The executive considers each designated station as a distinct, valid user channel to the same common program. On the executive level of communication, each joined station can operate almost individually. "Almost", because certain obvious restrictions are made upon the joined stations to prevent disruption of group operation by an individual. Such restrictions include preventing a change of program status to a non-operating mode, i.e. debugging mode, stop or quit status. Individual stations may remove only themselves from the group by quitting; only the originating station may stop or terminate the program by executive action.

Joining service primarily provides multi-user input service to an object program; output messages are directly addressed to specific station numbers. This means that any object program may be used to generate output messages to *any* user station (one at a time).

The most obvious problem for "joined" stations, namely the proper sequencing of the multiple user access to the common program, was not a very difficult one, since the scheduling logic was based on a fixed, channel table. However, human errors, as usual, predominate in an on-line system, and special precautions on joining had to be made.

The originating station has to be prevented from joining a stranger against his will, that is, an independent user might suddenly find himself joined with an experimental game that he really has no part of. Thus, only inactive (no programs loaded) stations may be joined. As mentioned earlier, any individual joined station (except the originator) must not issue executive commands which will disrupt the remaining group. Those commands which can affect the operating program status, such as stop, quit, and debugging commands, are only honored from the originating station. A joined station can stop or quit his individual participation any time. An additional requirement showed up when the wrong station became joined accidentally, or if participation was to be terminated for any joined station. If the station was remotely located and if the station's user would not or could not effect termination (quit), the joined status could not be undone. For this reason, the UNJOIN command was added. (The originating station, obviously, may not be unjoined.)

The JOIN function has proved quite valuable for various experimental exercises using groups of live subjects. In several cases, the game playing group involved remote user participation utilizing common carrier connections. Supplementary direct conversation was provided through the DIAL function described below.

DIAL

The DIAL function permits station-to-station communication through an executive command. One or more stations may be specified as addressees and a message of limited size (total length of one teletype line, including command and addressees) will be delivered immediately by the executive. Delivery is made as soon as the receiving station's output buffer is ready. All stations may be dialed simultaneously (by the computer operator) for a public broadcast.

The DIAL capacity was very essential to TSS in its formative stage, since it was (and, to some extent, still is) a tape dependent system before the disk was installed. Station communication with the computer operators was quite heavy, and the DIAL command proved to be a significant asset.

The main problems which the DIAL function imposed, included interference with user program output, restriction on message lengths, and the necessity of formally giving the DIAL command each time. Very often, formatted program output would suddenly be interrupted by a DIAL message from another station. This could be most annoying to the on-line user, especially if it were a "wrong number". Because of the limit on DIAL message lengths, messages had to be segmented, and, frequently, some trailing characters of the mesage were truncated. Improvements to DIAL service were contingent on acquiring more storage (disc) for executive services, and such improvements are being made.

LINK

The linking capability enables two stations to act in concert with an object program, where the program has normal access to a single user. Any station may LINK to another station that is not currently linked. A LINK notice appears at both stations, stating who is linked to whom. Both stations are now in communication with the object program, if any, that was operating for the user that was linked to ("linkee"). All inputs and outputs to and from the system are immediately seen at both stations, and a "conversation" mode is provided for unlimited, direct communication between the linked stations.

The LINK operation became at the same time a most interesting function and an interesting problem. In its simple form, the LINK function has been used for remote demonstrations, monitoring of various on-line operations, joint debugging, and for consultation and teaching services to new, remote users. Perhaps one of the most unexpected payoffs from the LINK feature was the ability for a user to operate several programs in parallel. By linking to unused channels, the user could initiate an operation, unlink, and repeat this action on another channel. He may then periodically check any of his linked operations at will. Obviously, such an approach is highly useful for production tasks and becomes an aid to the computer operator for background work.

The LINK function is a public one, available to any user station. The problem of unexpected output interruption (as for DIAL) was present and complicated by the fact that the "intruder" was now able to lay hands on the user's program. The linking station could issue a termination (quit) and thereby destroy the "linkee's" entire operation. Then too, cases occurred where sensitive information was being output and, by using the LINK function, a stranger could see the restricted data. Although either of the linked stations could unlink, they could also interfere with and prevent that action from taking place. It was no wonder then, that during a local demonstration that was to be monitored remotely, as soon as the LINK notice appeared on the demonstrator's Teletype, he quickly panicked and appealed to be unlinked because his visitors were arriving at any moment.

Another problem often occurred in using the LINK facility for teaching. It became necessary for the neophyte to avoid interfering, accidentally or otherwise, with the consultant's input actions. Since only unenforceable, procedural rules could be provided, this was not always possible. Something more positive, that would render one of the linked stations passive (in terms of input), was required.

As a result of these various problems with linking, variations were designed for this function. For simple monitoring (watching) of another station's operation, a "secret" link (SLINK) would be used. This did not cause any disruptive printout on the monitored station's teletype. Furthermore, no inputs from the monitoring station could be directed to the monitored program or seen at the monitored teletype. In effect, this form of linking involved only output monitoring.

To cope with the problem of disarming any interference from a linked station, a "passive" link (PLINK) could be employed. This feature prohibited all program or system inputs from the monitored station from being honored, although all inputs (and outputs) from the monitoring station could be seen. This form of control would be enabled and disabled at will by the monitor station.

Finally, a STOP LINK command to the system would prohibit any station from being able to link to a user's station in any fashion. This defense would be required in the face of a potential invasion of privacy by linking.

All of the above three functions require administrative approval before being activated by the time saving executive, and thus such commands would be forwarded to an administrative channel (i.e. the computer operator) for permission.

A remaining LINK feature which may be implemented in the future, is to link more than two stations together and impose a certain amount of procedural control over such group activity. Programs may behave (i.e. be debugged) but the human operators may not. In the real world of human fallibility, it is essential to provide proper controls, idiot proofing, escape hatches, etc., for the effective employment of on-line computer systems.

CONCLUSION

The on-line approach provided by time sharing will accelerate the usefulness of computers many fold. Direct MAN-MACHINE inter-

action and pooling of software services will make a computer much more flexible and responsive to the needs of individual users. However, it is important to consider the communication and interaction needs of groups of users with computer programs, to realize more fully the benefits of computer assistance. By making provisions for interaction of MAN-MAN, MEN-MACHINE, etc., the human relationship with computers will not be unnaturally restricted and greater rapport between man and machine elements in computer-based systems will ensue.

REFERENCES

[1]Charles W. Adams, "Cottage Computing", *Datamation*, Vol. 7, Number 10, October 1961.

[2]Fernando J. Corbato, Marjorie Merwin-Daggett, and Robert C. Daley, "An Experimental Time Sharing System", Computer Center, MIT, APTPS *SJCC*, 1962, pp. 334-335.

[3]D. C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework," Stanford Research Institute, Report AFOSR-3223, October 1962.

[4]E. Fredkin, "The Time Sharing of Computers", *Computers and Automation*, Vol. 12, November 1963, pp. 12-20.

[5]J. C. R. Licklider, and Welden E. Clark, "On-Line Man-Computer Communication", *Proceedings SJCC*, 1962.

[6]Calvin N. Mooers, "The Reactive Typewriter Program", *Communications of the ACM*, Vol. 6, Number 1, January 1963.

[7]Arthur M. Rosenberg, (ed.), "Command Research Laboratory User's Guide", System Development Corporation, TM-1354, 1963.

[8]D. T. Ross, and J. E. Ward, "Picture and Push-button Languages", MIT Electronic Systems Laboratory, Interim Report 8436-IR-1, January 1961.

[9]I. E. Sutherland "Sketchpad" A Man-Machine Graphical Communication System", *Proceedings SJCC* 1963.

[10]J. I. Schwartz, E. Coffman, C. Weissman, "A General-Purpose Time-Sharing System", System Development Corporation, SP-1499, 29 April 1964.

[11]Herbert M. Teager, "The Marriage of On-Line Human Decision with Computer Programs", MIT, *Naval Res. Logist, Quart.*, December 1960, pp. 379-383.

Massachusetts Institute of Technology
Project MAC Summer Study Memoranda

[1]D. C. Engelbart, "A Form of Group Interaction Easily Available on CTSS", MAC-M-96 (S63), August 16, 1963.

[2]D. C. Engelbart, "Results of an Experiment in Man-Machine Interaction", MAC-M-94 (S63), August 16, 1963.

[3]U. Neisser, "Parallel Consoles", MAC-M-67 (S63), August 6, 1963.

[4]Arthur M. Rosenberg, "Group Interaction Conferencing and Consulting in the Time Sharing System", MAC-M-45 (S63), July 29, 1963.

IV

METHODS

Dr. Herbert F. Mitchell, Jr.*

Message Switching Plus

INTRODUCTION

THE TERM "message switching", when referring to an application of a digital computer, has been almost exclusively related to the switching (and related functions) of telegraph messages. The message switching function has crept into many fields of computer usage—in fact, wherever communications circuits are terminated at a digital computer. This fact makes it mandatory that we enlarge our concept of message switching to include these other applications. In particular, we need to see what implications on hardware and software design and standards this enlarged view may have.

The marriage of communications and computers is taking many forms. Among the more easily recognizable are: (1) query networks (reservations, stock market statistics, credit, library reference, any inventory situation); (2) data collection networks (personnel "time" data, move station data on assembly lines, instrumentation data in a process control environment, location of shipments in transportation systems); (3) time sharing of computers among remote users (Project-MAC type); (4) man-console-computer association in the solution of unstructured problems (the so-called on-line computing technique); (5) computer-to-computer hook-ups for sharing load or dividing processing functions; (6) remote distribution of the results of automatic data processing; and (7) the switching of telegraph messages.

Each of these many forms of communications involving computers has its own peculiarities, and yet there are many functions and features which are common to several or to all. It would appear timely to assess these features for their commonality and their peculiarities in order to guide those responsible for the development of new hardware, new software, and new techniques in their search for efficient *and* compatible solutions.

This paper describes a computer controlled communications network which contains all elements necessary to serve the common on-line applications involving computers and communications, and can time share its facilities among any combination of such applications.

"UNIVERSAL" COMPUTER-CONTROLLED COMMUNICATIONS NETWORK

The parent network from which any of the above enumerated systems may be derived is simple to describe. A digital computer is provided with a multiplicity of input/output data channels, some of which terminate in communications lines. Other terminations include a data bank, direct input devices, direct output devices, consoles, and other computers. The communications lines, in turn, may terminate at input devices, output devices, (consoles, telegraph stations), another computer's communications terminal, or another telegraph central. **See Figure 1.**

To the best of the author's knowledge, no single computer installation to date involves all elements of this parent network. Each application enumerated above is satisfied with a subset of these elements. The question may properly be asked if there is any need to con-

*Vice President, Advance Systems Development
The Bunker-Ramo Corporation.

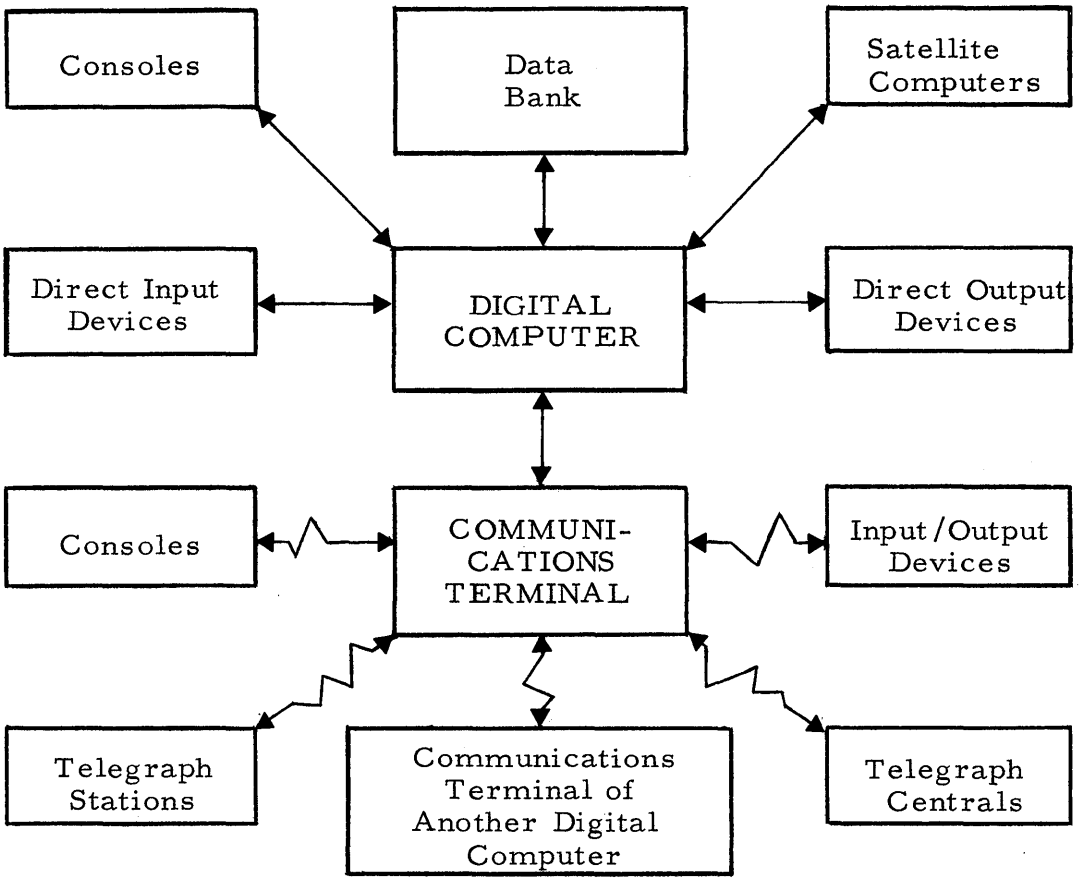


FIGURE 1

"UNIVERSAL" COMPUTER-CONTROLLED COMMUNICATIONS NETWORK

sider the parent network as one for which a demand will arise.

As the techniques and hardware for utilizing a digital computer from a remote location become more capable and less costly—and both trends are strong today—the pressure to use existing facilities more efficiently will grow. For example, much of the telegraph traffic handled by the common carriers is carried over voice-grade lines, but at telegraph speeds. Yet in many systems, the line costs are the largest single element in the user's budget. If the user could terminate these long lines in a computer already present for the other necessary functions, the traffic rate over the high cost lines could be increased 30 fold with only a three times increase in tariff. Users employing multi-circuit facilities at reduced tariffs (such as Telpak) could realize such increase in traffic carrying capacity at no increase in cost.

The need to time share the communications lines among several functions within an organization is developing rapidly—not only for economic reasons, but to capture vital information for use in several application areas. Such time sharing becomes feasible only if the full network of **Figure 1** can be implemented. The problems to be solved before these goals can be reached are many, but not insurmountable.

MESSAGE SWITCHING IN THE "UNIVERSAL" NETWORK

Messages may originate from many types of input devices—instruments, radio receivers, A-D converters, keyboard, paper-tape readers, card readers, magnetic tape readers, and computers. The network must control the initiation of the message, or be able to respond to an arbitrary initiation. The connecting communications line may be simplex, half-duplex, or full-duplex. The simplex line permits transmission in one direction only—out-station to computer or computer to out-station. The full-duplex line contains two independent simplex circuits, one for each direction of transmission. The half-duplex line contains only one circuit, but the direction of transmission is reversible. Since both transmitting and receiv-

ing stations must reverse direction simultaneously, some means of controlling the reversals must be provided in *either* out-station *or* computer.

In one type of controlled operation (such as telegraph out-stations on a "party" line), the computer is idle in the transmit mode, whereas the out-station is idle in the receive mode. Periodically, the computer sends a polling symbol unique to the out-station and reverses its mode. Upon recognizing its "name" symbol, the out-station reverses its mode, if operational, and responds within a given period of time. If the out-station is operational but has no traffic to transmit, it must respond to the polling symbol with a unique no-traffic symbol. If the response is a message, the first message character or start-of-message symbol must be different from the no-traffic symbol. There must be a unique end-of-message symbol. The line is held by the polled out-station until (a) it generates a no-traffic symbol, (b) it generates an end-of-message symbol, or (c) it generates no symbol during the allowed responding time. The line reverts to the control of the computer upon the occurrence of one of the situations (a), (b), or (c), and remains under computer control until the computer transmits another polling symbol. Many out-stations may share the same line, but only one may transmit at a time.

In a second type of controlled operation (such as query networks using "party" lines), the computer is idle in the receive mode and the out-station is idle in the transmit mode. A local controller performs the function analogous to polling, which permits each connected out-station, in turn, to gain access to the line when it is seeking to transmit. Once the access has been granted, the controller locks on to the selected out-station until the return message from the computer to that out-station has been completed, whereupon both computer and out-station revert to their respective idle modes.

In the uncontrolled mode, the input station may initiate a message at any time. Operationally, this mode is equivalent to that state of the controlled mode between transmission of a polling symbol and the occurrence of a terminating situation (a), (b), or (c). It is obvious that only one out-station may be allowed this mode of operation on any line, and the computer has no control (on this line) of the input station.

Where traffic requirements exceed the capacity of one line, or where message lengths allow one input station to eclipse another of higher priority, it may be advantageous to route polling symbols over one line connected to many out-stations, whereas the traffic to or from these stations is routed over several lines.

To prevent undue periods of idleness for a line, it is desirable for the computer to respond to a transmission (other than the no-traffic symbol) soon after termination of that transmission. If the response is a return message, it may be desirable to hold the line open until the return message is ready. In other cases, the computer response is simply a "received OK" or "not received OK" symbol. In the latter situation, the computer does nothing with the message, expecting a retransmission.

If the connecting communications line is a single circuit (simplex or half-duplex), the input station can pre-empt the line for indeterminate periods, as far as computer control is concerned. During such periods, traffic may be entered by the station which has to be received, recognized, and appropriately dealt with by the computer. The computer must be able, in general, to ascertain: (a) the identity of the originator, (b) the start of the message, (c) the end of the message, and (d) that portion of the message which will determine the nature of the computer's handling of the message.

In establishing identity of the originator, the line connection at the computer will suffice if only one originator is possible on this line. Otherwise, an originator identity symbol must be present in a recognizable place in the message. As far as possible, out-station identifying symbols should be automatically generated and always present. Thus responses and return messages can be assured of return to the originating input station.

A start-of-message symbol is not mandatory, since the first character to follow an end-of-message symbol, a polling symbol, or a start-up condition can be assumed to be the first character of a new message (except for the unique no-traffic symbol, if used). Such a symbol is useful, however, to guard against accidental or unintentional transmissions, and to expedite synchronization following a line outage.

The end-of-message symbol is mandatory in controlled-mode operation, and highly desirable, if not mandatory, in the uncontrolled

mode. An exception is permitted if all message traffic has the same length of message, in which case the end of message is implicit.

From the message-switching standpoint, the most vital part of the message is that which contains the characters which determine the computer's handling. These may be identified by position, such as the first characters of the message, or be a set delineated by special start-of-address and/or end-of-address symbols. If explicitly addressed messages are combined with implicitly addressed messages, distinctive start-of-message symbols are normally required. A simple directory reference should suffice to categorize the message and initiate the appropriate action on the part of the computer. This action may take many forms, of which the following are illustrative:

1. Dispatch the message to one or more outgoing lines, taking cognizance of indicated priority and executing all required message edit, audit, and statistical recording operations.
2. Store the message for later treatment.
3. Acknowledge receipt of the message to the originator.
4. Send rejection message to the originator.
5. Perform reference to data bank, using key compiled from "address" characters and/or directory entry.
6. Establish a status, depending upon the content of the message, the directory, and/or the data bank entry.
7. Generate one or more messages to known addresses and/or originator depending upon the resulting status determination and/or including the content of the data bank entry.
8. Initiate a compiler, assembler, or correction procedure upon designated program elements.
9. Retrieve and initiate a computation with a designated program.
10. Update a status record for all actions taken on this message.

At the destination end of the communications line, the receiving out-station may be the counterpart of some input station, or may be a "read only" station. If the latter, there must be a way for the computer to determine if the device is operational, such as its response to the receipt of the message. If the connected line is half-duplex under controlled operation, the out-station is normally in the receive mode when idle. However, in the idle state it is responsive only to its unique "name" symbol. Upon recognizing this, it prepares itself to receive and record the message, monitoring the character stream for the end-of-message symbol. When the end-of-message is recognized, the station places itself in the transmit mode, returns a "received OK" or "received not OK" symbol, and then reverts to the receive-idle mode. The computer, upon sending the end-of-message symbol, places itself in the receive mode until the station response is recognized, or until a time-out occurs. If the response is "received not OK", the computer is normally programmed to repeat the message transmission at least once, before indicating trouble.

If the out-station permits both input and output, the line must be either half or full-duplex. If half-duplex, one of the two types of controlled modes must be used. If the computer controls the idle line, it may transmit either a polling symbol to a particular out-station in turn, or a directed message to a particular out-station. The out-station behaves as an input device if it recognizes its polling code, or as an output device if it recognizes its directed code. If the local controller controls the idle line, the line must be held for the selected device until the computer returns its answer message.

CONCLUSIONS

It should be apparent from the foregoing that there needs to be a fairly high degree of compatibility in the hardware, software, and techniques of operating a "universal" computer-controlled communications network. Certain restrictions are immediately apparent. A given communications line must operate in a specific mode at a specific speed, and use a homogeneous character set. If the line is to be shared by stations of several kinds, the response characteristics of all connected stations to a given symbol must be identical or mutually exclusive.

The computer program can easily be made to be "line sensitive". That is, it may expect telegraph message traffic over one line, special key-set messages over another, and arbitrary bit patterns over still another. However, if full advantage is to be taken of leased line sharing, and particularly if several kinds of

message traffic are to be routed over public lines, it becomes important to standardize on character set, control symbols, and operational mode. When a message has been received by the computer, it must contain all features which will enable it to be processed, dispatched, and the appropriate response made to the originator, depending upon its content alone. Input line identification should be replaced by identification symbols generated at point of origin, or en route. The handling of the message should be independent of the manner in which the message reaches the computer.

While the computer is capable of translating one character set into another, and responding to several sets of symbols with the same meaning, such requirements lead to more complex programs, longer throughput time, larger memory requirements, and greater chances of error, both programming and operational. The system should be designed to operate on a given character set (ASCII is the official standard), one set of internally generated control symbols, and lines of as few speeds and modes as possible. Where terminal devices are incompatible with the above, translation hardware should be provided at the interface to the system. In the interests of system maintainability, record keeping, and updating, all anomolous conditions unavoidably present at the outset should be considered temporary, and system design should be such that standards will eventually be observed in all areas of the system—hardware, software, and operational techniques. The increase in system performance, ease of updating, and smoothness of operation that can result from such observance of standards will more than repay the investment.

Donn B. Parker*

Graphical Communication in an On-Line System

INTRODUCTION

PROCESSING GRAPHICAL data is a major application of digital computers. This data is usually treated in a highly stylized fashion integrated into specific problem solutions. Treating this data in a digitized "picture" type representation and making it the subject of communication between a man and a computer in the man's own real-time frame opens a new dimension of applying computers in solving design problems. Presented here are the significant parts of an idealized graphic system which, if implemented, should make this possible in an economical, sophisticated, and practical way. This system should prove to be a useful design and problem solving tool. The basic hardware and software provides a basic drawing capability but only as a means to achieve real design objectives. An interface with application programs and the capability to label and treat the data as "things", not merely as line segments, provides the design and problem-solving capabilities.

OTHER SYSTEMS AND BACKGROUND

Already implemented graphic systems exist which meet some of the goals and purposes of this proposed system in various degrees. Such systems as the General Motors Research DAC1 System[1] and the RAND Table[2] are used for research and development with very basic and general software. The MIT Sketchpad[3] has taken an interesting approach for making drawings and doing design.

The proposed system described here has been influenced by the above systems and also by systems developed by ITEK Corporation and Charles Adams Associates[4].

GOALS AND EXAMPLE OF USE

The goals of this graphic system in providing a practical and powerful tool for general design activity are economy, ease of use, and availability. The console must be comfortable to use, an obvious advantage over other methods and tools, and minimize human stress. The basic features, besides interfacing with design application programs, must provide powerful and flexible drawing capability in the sense of formal engineering drawing, revising drawings, engineering sketching, and a variety of other graphic uses some of which are as yet not conceived.

An example of its use might consist of the following. One of several consoles, connected to a central computer, is installed in an office shared by four design engineers. Information from a master file relative to their project is available for display by console request. Each engineer could make his own formal engineering drawings directly on the console. He rarely needs a drawing on paper, and when he does, discards it after use. He still makes sketches at his desk with paper and pencil but quickly redraws the useful ones on the console cathode ray tube (CRT) during the design process and saves information in his own working file in the mass storage of the computer. He confers with an engineer at another console by telephone while they are both viewing a drawing and making design

*Staff Specialist, Control Data Corporation.

changes. Many computer programs requested from the consoles aid them in their design and analysis.

HARDWARE AND SOFTWARE ORGANIZATION

The console for this system consists of three principal elements—a large cathode ray tube (CRT), a keyboard, and a light pen. A large, relatively flat faced, circular CRT, with resolution for one million discrete beam positions, is set into a console in a hinged frame so that it may be positioned at a variety of angles in the manner of a drawing board. The CRT is protected by a plate of glass. The console is convertible to desk or drafting table heights.

A light pen of normal pen size, with a microswitch on its shank to turn it on and off, is connected to a photomultiplier tube with a fiber optics cable. The concept of the light pen and its operation are well known[5, 6].

The keyboard is movable and has a magnetic backing. It may be set on a table surface to the left or the right of the user or adjacent to the CRT. The objective of its design is to keep the mechanical features to a minimum thereby reducing the cost and increasing the potential reliability of the system. Two of the buttons on the keyboard operate in parallel; one on each side of the keyboard, for symmetric availability of a major function. One button operates in parallel with the light pen microswitch. The number of buttons is minimized to keep operation of the console simple. Each button activates an interrupt line to the computer. Each is spring-loaded and non-latching. Assignment of button functions is software controlled, and removable tabs explain their use.

Other possible mechanical features such as foot pedals, alphameric keyboard or other keys are not included since they complicate console operation and increase console costs. Also, adequate features are easily introduced by software on the CRT. Reversal of these decisions could result from future experience.

Several consoles are driven by a controller which contains a display memory, light pen interrupt circuitry, and optional character generation circuitry. The display memory stores the CRT beam driving instructions, sequencing instructions, and console designation instructions. The beam instructions are incremental in X and Y from the last beam position, or they instruct beam movement to an absolute position. The amount of display memory is assigned to each console as needed. The rate of generation is 30 to 40 frames per second to produce a relatively flicker free picture. The amount of beam travel per frame need not be excessive because of the limited capability of the user to assimilate and make use of detailed visual information and because of flexibility of software to compensate for this limitation. The display memory has its most important function as a buffer to give the consoles a complete off-line status except when an external interrupt initiates changes in the display. Capability to display from computer memory is also required for high speed display changes, such as light pen tracking.

One or more controllers may be used in a computer configuration, depending on the capacity and speed. The entire computer capability may be used to serve consoles, or the computer graphics capability may be time shared with other peripheral devices and batch mode operation. The computer requirements in speed, memory capacity, and secondary random access storage capacity are considerable in view of the storage needed for complex digitized drawings and the major programmed features to be described.

The graphics software consists of the resident executive to process the console interrupts, basic subroutine execution, display generating routines, application interface, applications programs, and library sub-routines. The major aspects of the software will be presented from the functional point of view in terms of console features and capabilities.

The graphic console features, when specified in all detail, constitute a language which could be presented in a formal way. This is a manual language in contrast to spoken or programming languages. As graphics systems are developed and used, several such manual languages will emerge along with familiar problems of attempts at standardization. It is hoped that experience and insight already gained in language standardization will minimize this problem. Formation of users' groups, communication of graphic construction algorithms, and an Association for Computing Machinery special interest group are already foreseen for the near future.

BASIC FEATURES

A concept of precise, literal drawing is employed. Graphics are described in an analy-

tic geometry representation, and topological freedoms are not allowed, with one exception. When the pen is pointing close to a graphic element or to one of its parameters, it is assumed that it is pointing at that element or parameter in an exact analytic geometric sense. If the pen is pointing close to the end of a line segment, the computer assumes it is not pointing at the line but at the exact end point represented in the computer by coordinate values to the full precision capability of the computer. This is one of the most significant features of the system. Although the construction of graphics is instigated and viewed by a man at very low precision, the computer completes the construction and saves it at a precision far beyond that of the man's and the console equipment's capabilities.

The basic units of data are X and Y coordinate values, lengths, radii, angles, address pointers, and alphameric characters. These are combined with geometric, text, logical, and display codes to form entities such as straight line segment, circle, text, and group **(see Table 1)**. A distinction is made between a point entity which is a temporary parameter for graphic construction and the dot entity which is a point entity formally introduced as part of the drawing.

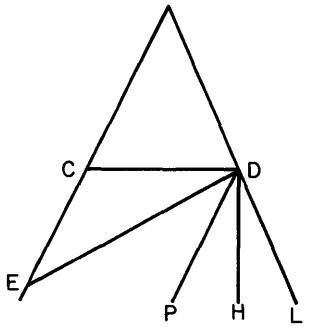
The CRT display surface is divided into two types of regions under control of the basic software, but it can be changed by application programs. The working region is a central rectangular or circular area outlined by a displayed frame entity. Graphic construction is carried out only in this region. The second is called the control region and it constitutes the remainder of the display surface. It is used to display light button and light register entities also under control of the basic software and changeable by application programs.

TABLE 1

LIGHT REGISTERS

X_1	$Temporary_1$	Label Display
Y_1	$Temporary_2$	Label Input
X_2	Base Angle	Message Register
Y_2	$Angle_1$	X Frame Location
$Length_1$	$Angle_2$	Y Frame Location
$Length_2$	Zoom Index	

A light button is a displayed group of alphameric and/or geometric entities which when selected, by pointing the light pen at any part of it, causes an ON condition. For example, the type of geometric entity desired for construction is indicated by selecting the appropriate line or dot representing a conic section displayed in a two-dimensional projection of a cone. **(See Figure 1).**



D	Dot
L	Straight Line Segment
C	Circle
E	Ellipse
P	Parabola
H	Hyperbola

FIGURE 1

GEOMETRIC LIGHT BUTTON AND CODE MEANINGS

A light register entity is a blank space for display of a number, label, or message with its name displayed to the left. The light pen is used to pick a register for use by pointing at any part of its displayed name. Information may be placed in a light register by either the user or the computer. Light buttons control inter-register transfers, filling, clearing, and locking. Message registers contain alphameric information for man-computer communication. Light registers may be added, deleted, modified, or moved by application programs, since they and their contents are part of the data list in the computer. **Table 1** lists the light registers provided by the basic software and shows the extensiveness of register use and insight into console features.

LIGHT PEN PICKING

The selection of displayed entities as parameters for use in further construction is the most frequently performed function and should be the simplest to perform. The light

pen is pointed at an entity, the microswitch is turned on and then off, and the entity is picked. Entities in the working region and light registers in the control region may be picked. A successful pick is indicated by momentary light intensification of the entity picked followed by the display of a character next to it or surrounding it (in the case of a dot). A light button is selected (as opposed to picked) by the light pen to cause an action to take place. Momentary light intensification of the light button shows computer acknowledgement of the selection. When an entity is picked, displayable parameters of the entity such as the center of a circle or the focus of a parabolic arc are also displayed as small, low intensity crosses.

A large number of entities may be picked before they are used in a last-in-first-used order. Higher levels of picking are performed by subsequent picking of already picked items. Picking a circle which has already been picked causes the group of entities which contains the circle to be picked as a group (not as individual entities). The circle remains separately picked. The erase and other functions will change the status of items. **(See Table 2).**

ENTITY TYPES

An entity may be created in many ways, but only the basic parameters are used to store and define the entity. These parameters are chosen on the basis of minimizing storage requirements and making equation representations easy to derive. The entity types available in the system and the basic parameters which describe them are listed in **Table 3.**

TABLE 3

ENTITY TYPES

Geometric	
Dot	X, Y.
Straight Line Segment	X_1, Y_1; X_2, Y_2.
Horizontal Line Segment	X_1, Y_1; X_2.
Vertical Line Segment	X_1, Y_1; Y_2.
Circle	X, Y (center); R (radius).
Circle Arc	Circle parameters; X_1, Y_1. (beginning point, counter-clockwise); X_2, Y_2 (end point).
Ellipse	X, Y (center); A (semi-major axis length); B (semi-minor axis length); L (angle of major axis).
Ellipse Arc	Ellipse Parameters; X_1, Y_1; X_2, Y_2 (circle arc sense).
Parabolic Arc	X, Y (vertex); L (axis angle); F (focal distance); X_1, Y_1, X_2, Y_2 (circle arc sense).
Hyperbolic Arc	X, Y (center point); A transverse radius); B (conjugate radius); L (angle of major axis); X_1, Y_1, X_2, Y_2 (circle arc sense).
Straight Line Segment String	X_0, Y_0, X_1, Y_1, ... X_n, Y_n.
Text	X, Y (lower left corner of first character); string of BCD characters.
Group	List of pointer addresses of group members.
Control Region	Control Region location.
Register	Register name; contents.
Light Button	Light button name.
Frame	X, Y (center point).
Application	Parameters assigned by application programs.

TABLE 2

STATES OF ENTITIES

Erased	No longer exists.
Non-Displayable	Exists in the computer data list but is not displayable.
Displayable	Exists and is available for display.
Parameter-Picked	Is currently displayed, and one of its parameters is in picked status.
Picked	Is currently displayed as picked and is in picked status.
Group Picked	Is currently displayed as picked, is not itself necessarily picked, but the group or higher level group of which it is a member is picked.

The geometric entities are limited to planar conics. Higher degree curves may be described by fitted conic arcs or by the straight line segment string. Three-dimensional graphics may be described by their two-dimensional projections.

Several line intensities and standard line types may be selected by light buttons. Several extra line type buttons are left unassigned to provide line types for the user or application program designation. Other light buttons provide for nondisplay and redisplay by line type or line intensity. For example, a drawing may be studied with all center lines or all dimension lines temporarily non-displayed.

DATA FORMAT

The data format of graphics in the computer is organized to optimize the compromise between data compactness, ease of use, and flexibility. Some of the n-Element Component and Plex ideas of Ross and Rodriguez[7] are used. The active data list for a single console is stored in a combination of high speed memory and secondary storage in a relative addressing, blocked structure.

An entity is headed by a key field identifying the type of entity, line type, line intensity and other parameters defining its current status. The next field is a label with contents specified by the console user or application program. Its many uses include attaching design significance to the entity or superimposing a secondary list structure. The third field is a pointer which contains the relative address of the parent entity if one exists (**See Figure 2**). The remaining fields contain the parameters listed in **Table 4**.

The data structure makes it possible to locate a group in a tree structure from knowing a member of the group or from knowing the label attached to a group entity. Entities may be grouped by the user at the console or by application programs. There are three methods of grouping entities for various uses: assigning common line types or intensities, picking entities for a group entity, and attaching labels to entities. Picking groups is facilitated by a set of light buttons providing for top group picking or picking in incremental steps.

A further breakdown of the data structure is possible by separating the X and Y values in a point table and providing pointers in an entity array to locate its point parameters. There seems to be no advantage in doing this in a two-dimensional system either in storage savings or program simplification.

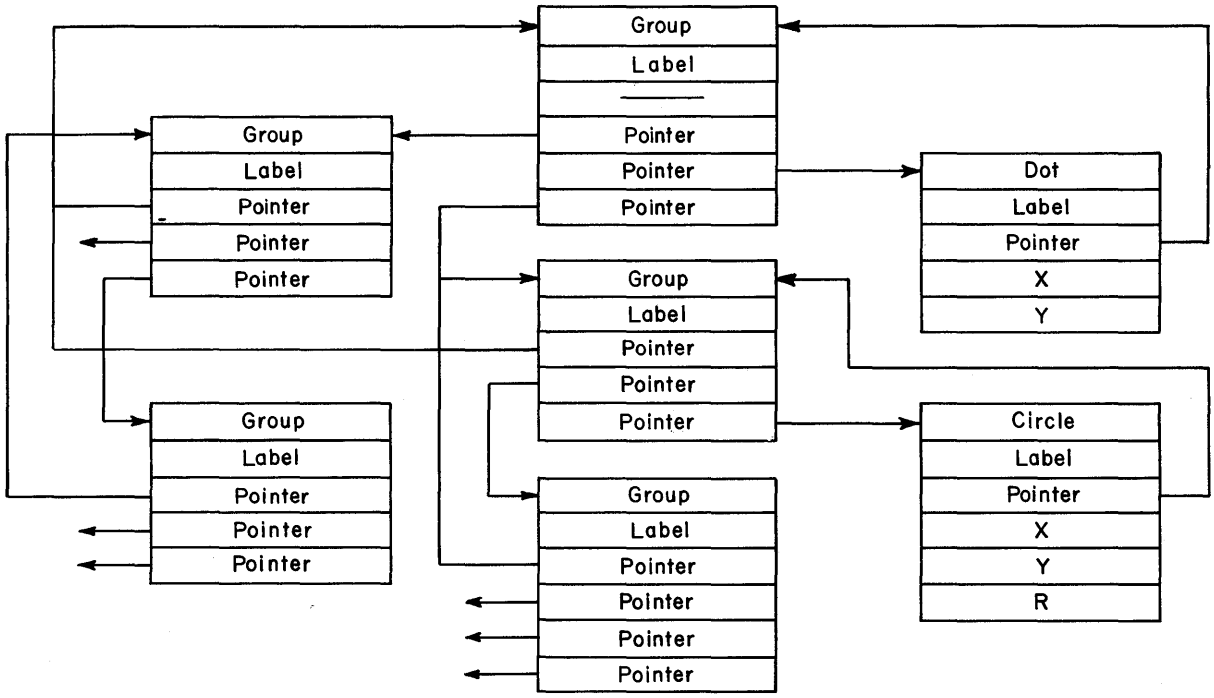


FIGURE 2 TREE STRUCTURE OF GROUPS

ORDER DEPENDENCE

The console features, especially order dependence to be described here, are sophisticated and require a significant amount of learning and experience for optimum use. The effort required to identify and correct errors is minimized. A programmed instruction application program is specified to teach new users and is available at any time for review instruction during console use. The graphic capability and software controlled buttons and registers render the system ideal for this technique.

A concept of order dependent parameter specification is employed in ambiguous cases to impart necessary meanings. The order in which parameters are picked determines how they are to be used when more than one way is possible. If two points are picked and the construction of a circle is requested, the first is the center and the second is a locus point. If the user forgets the order imposed meaning, he guesses and if wrong, erases and starts again. This avoids the necessity for parameter descriptor buttons for center, locus, end point, focus, vertex, radius, and axis. **Table 4** shows all the allowed combinations and ordering of parameters to construct a circle. The numbers indicate order dependence, and X's indicate order independence. Combinations are read vertically.

TABLE 4

CIRCLE CONSTRUCTION PARAMETERS

Point or X, Y Registers	1	X	X	X			
Point or X, Y Registers	2	X					
Point or X, Y Registers		X					
Length Register			X			X	
Straight Line Segment				X		X	X
Straight Line Segment						X	X
Straight Line Segment							X

It is impossible to specify too many parameters for a construction because only the first set of meaningful parameters encountered are used in a last-in-first-used basis. Based on the first acceptable parameter encountered in this order, all other non-meaningful parameters are ignored. **Table 5** shows several push-down-from-the-top lists for the circle construction. An asterisk indicates the chosen parameters.

TABLE 5

PUSH DOWN PARAMETER LISTS FOR CIRCLE CONSTRUCTION

*Point	(2)*Point	*Point	*Line
*Point	(1)*Point	*Line	*Point
*Point	Line	Point	Point
*Line	*Line	*Point	*Line
Line	*Point	*Line	*Line
*Point	Line	Line	*Line
*Point	*L. Register	*Line	*L. Register
*L. Register	*Point	L. Register	Line
_____	_____	*Point	*Point

If there are not enough meaningful parameters given when the construction is specified by a light button, "MORE" appears in the message register. Construction proceeds when enough parameters have been picked. Thus, it does not matter when the construction button is selected relative to the parameter picking in this case.

Entities are constructed independently of the limiting frame of the working region. Thus, an entity can be constructed without appearing in the working region or it may only partially appear. An exception is, for open conics (hyperbola and parabola) only the arcs which lie inside the working region are created. Generally, arcs are constructed by creating the whole conic (in the working region) and then picking the conic and end points as parameters to the arc construction.

When three intersecting lines are used as parameters to construct a circle, several choices are possible: the inscribed circle tangent to the lines, the three exterior circles tangent to the lines, and the circle with the three line intersections on its locus. When a construction has been requested, one of the possible cases is displayed. The user must press the accept button to make the entity permanent, press the reject button to make it disappear, or press the alternate button to display another case for acceptance or rejection. Only one case may be accepted for any one construction procedure.

TRANSFORMATIONS AND CONSTRAINTS

Transformations provided by the basic software and directly available to the console user

and application programs are: similitude (uniform compression or expansion on a given point), translation, and the linear, orthogonal transformations rotate and reflect. This provides the user with the facility to change the scale of drawings or parts of drawings. Entities and groups of entities may be moved, rotated, and reflected. Only one half of a symmetric drawing need be drawn; and with the selection of a button, it may be reflected to produce the completed drawing.

Semi-precision drawing can be done by imposing constraints on freehand light pen usage. With the horizontal hold button depressed, the user can pick the tracking cross in the working region, and the drawing point normally traveling with the cross is constrained to move in a straight line as the pen is moved at the angle specified in the base angle register. The vertical hold button and angle hold button work similarly, one normal to the base angle and the other at an angle specified by an angle register. The actual creation of the straight line is accomplished by selecting a second button and accepting or rejecting that which is displayed.

Animation is to be avoided when possible because of the computer time required to display the intermediate positions of entities as they are transformed. A transform causes an original form to disappear and reappear in its final state. A transform-a-copy operation retains the original form and the new one appears. Animation may be valuable in certain applications of dynamic design analysis such as mechanical linkage clearance study. This may be accomplished with application programs.

DRAWING SURFACE

While considering the transformations and data structure, it is logical to introduce the concept of the drawing surface. This is the two-dimensional, imaginary surface on which all graphics are described. It is limited in size and detail by the range of the number representation and precision of the computer. In engineering design applications involving the drawing of objects, the drawing surface is virtually unlimited in size and precision when the computer has a 48 bit word, floating point number representation. The working region of the CRT is used to view parts or all of the graphics on the drawing surface just as a variable magnification glass might be moved over a map to view parts of it in detail. The zoom register, frame location registers, frame picking, and the translate transformation are used for this purpose.

The drawing surface can be looked upon as an imaginary two-dimensional computer storage where the contents of specified X, Y locations are points. The concept of drawing or subdrawing (groups of entities) relocatability is useful. Relocatability is simply defined by the translate transformation. A drawing may be stored in secondary storage such as magnetic tape and located anywhere on the drawing surface. After it is brought into computer memory, it may be moved to a convenient location. A drawing overlapping another is not prohibited. It is important to keep in mind the distinct difference between moving graphics on the drawing surface and moving the working surface frame of view on the drawing surface. Both actions are performed using the translate transformation, but the graphics to be moved are picked in the former case, and the working surface frame is picked in the latter case.

There is danger in loss of accuracy due to relocation. A drawing could be moved to a region of very high X and Y values and then returned to a region closer to the origin with complete loss of accuracy. Scaling could be of assistance, but it adds unnecessary complexity in most cases which can be kept under control by working close enough to the origin to maintain required accuracy. A non-linear drawing surface might solve this problem without significant additional complexity, but it is not considered in this paper. Overflowing the range of the number representation is indicated as an error in the message register. However, modulo treatment of the number range makes the drawing surface cylindrical in X and independently cylindrical in Y.

DRAWING SCALE

Drawing scale is normally an unnecessary consideration except when hard copy output is being prepared. Drawings are usually made in actual dimensions without any difficulty because of the vast range of the drawing surface as compared to the real world of objects to be drawn. The zoom function changes the scale of drawings, but only for visual purposes on a temporary basis which does not affect the data list. When a drawing is to be output in hard copy form, the user must change the

scale of auxiliary views with the similitude transformation as needed, add scaling notes to the drawing, and indicate the scale to be used by the output program.

PRINTING

Alphameric printing capability deserves an important place in a graphical system. Engineering drawings can contain very large volumes of notes, labels, and dimensions. Printing capability is provided in the form of a raster of alphamerics displayed in the control region. The procedure consists of picking a starting point, picking an angle if other than the base angle, and then employing a hunt-and-pick mode of character selection, all done with the light pen. Associated light buttons are also needed for margin control, font selection, and roll-up or roll-down carriage return. All characters are placed in a single text entity until an end text light button is selected. The alphameric raster is displayed in the control region only when selected by a keyboard button, and it disappears when the end text light button is selected. Erasure is done at the character level by the erase function. Numeric information is put into a picked light register from the right end and shifted left for each new digit or decimal point selected. The sign is entered independently in a fixed location.

There are several advantages to this type of software provided printing feature in contrast to a hardware keyboard. Advantages include economic savings in hardware, reduced maintenance problems, and increased console reliability. There is also an advantage in its use. All of the user's attention remains on the CRT and light pen. He does not have to put the pen down and turn to a keyboard. Printing and drawing graphics are highly intermixed in design making it an advantage for the two types of operations to be compatible.

DRAWING AIDS

Additional drawing aids are available in a separately maintained subroutine library. Light buttons are provided by the basic software to select them.

1. Display the intersection(s) of two entities.
2. Create a general conic arc given one of the following sets of five parameters: five points; four points and a tangent angle at one of the points; and three points and tangent angles at two of the points.
3. Create a curve of one or more conic arcs which is approximately parallel to a given conic and a distance $\pm D$ from it. The maximum error is displayed.
4. Compute the length of a conic or conic arc.
5. Display a point on a conic a given distance from a given point.
6. Compute the tangent angle at a point on a conic.
7. Display a straight line segment from a point to a conic and tangent to the conic.
8. Display a straight line segment tangent to two circles.
9. Construct a circular fillet given a radius and two line segments sharing a common end point.

Numeric results are also put into light registers. Order dependence is used extensively. When multiple results are possible, the user may select any or all by button control. Errors, input deficiencies, non-existence of results, and results off the working region or drawing surface are indicated in the message register.

CALCULATOR

An additional basic software feature is desk calculator-type capability. The basic operations including square root, multiply-add, and multiply-subtract, are handled by light buttons. The operand registers and result register are picked in an order-dependent sequence followed by selecting the operation which causes the action to be performed, all done with the light pen.

Additional functions including logarithmic, exponential powers, trigonometric, and frequently used constants are also easy to supply in light button form. The calculator set of light buttons is a good example of a control region feature which is not needed at all times, but can be displayed under control of a keyboard button when needed; thus conserving both control region space and CRT beam control instructions.

MACRO

One of the more annoying console operations is performing the same sequence of operations very frequently. The normal console features are micro in nature to provide flexibility. Application programs can be writ-

ten to provide automatic execution of sets of operations, but not as easily and quickly as sometimes needed. A drawing aid subroutine is provided with which the user may define macro console operations as he needs them from sequences of basic operations.

The sequence of operations to define and use a macro follows:

1. Pick the input parameters.
2. Select the macro definition button.
3. Select two characters and place in a box to be used as the macro button.
4. Step through the sequence of operations to be defined.
5. Select the macro definition button to terminate the definition. The box disappears leaving the macro button displayed.
6. The macro may now be used by picking the appropriate parameters and selecting the macro button.

Note that no parameters may be picked intermediate to the macro operation. When this is required, several macros must be defined and used in sequence and separated by parameter picking.

An example shows the usefulness of this feature. A copy of a template pattern such as a standard screw (represented by a group of entities in a standard template drawing on the drawing surface) is to be placed at the proper angle into the drawing under construction. A point at the base of a screw head is picked, the point of final position in the drawing is picked, and another point is picked such that the angle defined by the last two points is the desired angle for the screw centerline. The macro define button is selected, and the letters SC are placed in the macro button box in the control region. The operations performing the translate-a-copy are now performed. The first picked point is picked again to cause the whole group defining the screw to be picked, the X, Y registers containing the coordinate values of the second point are picked, the copy of the screw is moved to overlay the translate parameter points, the copy of the screw and the last two points are repicked, and the screw is rotated through the required angle. The macro define button is selected to complete the definition. A copy of any group of entities may now be moved and rotated by picking the three prescribed points in order.

INPUT AND OUTPUT

Input and output between secondary mass storage and the computer memory (drawing surface) for temporary and permanent storage is accomplished in the basic software by using a two file system. A permanent file of drawings is available to the user based on drawing titles. The file is updated only by an editing program from the second file called the working file. Each console user has his own working file in which are stored reference drawings, partially completed drawings and, possibly, drawings awaiting approval for transfer to the permanent file. A region of the drawing surface is reserved for a list of the names of the drawings in the work file. An arrow points to the name of the first or last selected drawing. The arrow may be moved by the light pen to a name of a drawing, selection of a light button causes the drawing to be deleted or placed in the computer (on the drawing surface). New names are added automatically as drawings are filed by the user. Hard copy of drawings may be produced on paper, microfilm, or other media by one of many available plotter or microfilm systems now on the market. Configurations may include systems on-line, off-line, and remote or adjacent to a console. One objective of the system is to minimize the desire and need for hard copy by making the consoles inexpensive, easy to use, and readily available.

APPLICATIONS INTERFACE

The most important software feature is the applications interface which makes the system a design tool and problem solver, not just a drafting machine. It consists of an executive application call routine and a set of FORTRAN subroutines. The purpose is to provide the data list and all the system features, except light pen tracking, in a form easy to use in FORTRAN application programs. All application program activity is carried out through the interface in a relatively non-changing environment to minimize the need for recoding programs when basic software changes are made. There are two distinct interface functions. One allows an application program to perform all functions available to a user, in the same way, except pen tracking which has no meaning to a program. A program may pick parameters, fill registers, select light buttons and, effectively,

push keyboard buttons. In addition, the program may inhibit the use of most console features, change their meanings or, in the case of control region features, change their positions and add new ones. Finally, a pause-until-console-action function provides an interrupt before or after action making the application program operable in the console user's real time. These features combine to provide a close working relationship between the user and computer, allowing the best characteristics of both to be brought to bear on problems. A master switch allows the user to turn off the application and return control to the console. This capability is unalterable by an application program.

A practical application program using this first function of the interface is one which records every action taken by the user in a time history. It produces a printed list naming each action and the time taken by it. This is most valuable in studying the man-machine interaction to make improvements in the system and to evaluate the skill and difficulties of the user.

The second interface function provides features more suitable to programming. These include routines to translate and include program-generated entities into the data list, to search the data list for entities based on any combination of characteristics, and to use the entity construction and manipulation facilities without going through the external console. For example, a call to the intersection subroutine with two entity relative addresses as parameters would return the intersection points as values assumed by a second pair of parameter names in the calling statement.

The system becomes a versatile tool with this capability. Many existing programs run in a batch mode computer operation can be fitted with preprocessors and postprocessors, thus, adapting them for on-line graphics capabilities.

Another important program executed outside the system provides a data structure and language translation for input of graphics to work files from handwritten coded sheets via punched cards.

Application programs are brought into memory from secondary storage by the executive application call routine at the request of the console user. An application program is executed until ended, until an interrupt for a high priority console request is received, until a pause statement is executed, or until a time limit expires. If control is to be returned to the application at a later time, it is kept in memory, or it is temporarily dumped into secondary storage, preserving its current state when memory space is needed.

APPLICATION EXAMPLE

A concise application demonstrating many of the system capabilities is the calculation of the area and centroid of a closed polyconic forming a simply or multiply connected domain and/or a system of dot-area centroids. The program steps and algorithm are as follows:

1. The randomly ordered, user picked entities describing the simply or multiply connected domain and/or the dots each followed by its associated area value are accepted.
2. A closure check is made. If lack of closure is discovered, the unconnected end points are displayed in small triangles and "MORE" appears in the message register. The user may press the reject button, in which case, the displayed triangles and message disappear and the program terminates. The user may also pick one or more entities which complete the closure, in which case, the triangles and message disappear and step 2 is repeated. If only a single closed conic has been chosen, the centroid is set to be the center point, and the area is calculated by standard formula. Go to step 8. Step 2 is ignored if all parameters are dot-area centroids.
3. Divide each double valued entity in X into two single valued entities. Discard vertical straight line segment entities, and divide straight line segment string entities into straight line segment entities.

$$Y=F_j(G,X,\overline{XY},\overline{S}),\ X_1 \leq X \leq X_2,$$
$$j=1, 2, \ldots p$$

are the p remaining entities where G is the geometric descriptor taking on one of the values: straight line segment, horizontal straight line segment, circle arc, ellipse arc, parabola arc, or hyperbola arc. $\overline{XY}$ is a set of fixed coordinate points, $\overline{S}$ is a set of scalars, and X_1 and X_2 the X coordinates of the end points. Thus a circular arc with center at h, k, radius R, and end points

at X_1, Y_1 and X_2, Y_2 ($X_1 < X_2$) would be represented by the equation

$$Y = k - \sqrt{R^2-(X-h)^2}, \; X_1 \leq X \leq X_2$$

where the sign of the radical was chosen negative because $X_1 < X_2$ and X_1, Y_1 is the beginning point of the arc in a counterclockwise direction.

$\left\{ X_c, \quad Y_c, \quad A_c \right\}_i$, $i=1, 2, \ldots, L$

are the dot-area centroids.

4. Divide the range of X, from smallest to largest encountered, among the end points of the p entities into subranges of X so that all end points of entities lie on subrange boundaries.

$$X_1 < X_2 < X_3 < \ldots < X_n,$$

X_i, $i=1, 2, \ldots, n$ are all the distinct coordinate values of end points of the p entities.

5. In each subrange, order, from the highest to lowest Y values, the entities included in that subrange by highest Y value each entity assumes in that subrange.

$$\left\{ F_{j_1}, F_{j_2}, \ldots, F_{j_m} \right\}_i,$$

$$\text{Max}(X_i, X_{i+1})F_{j_k} > \text{Max}(X_i, X_{i+1})F_{j_{k+1}}$$

for the m entities in the subrange X_i, X_{i+1} and for each i, $i=1, 2, \ldots, n-1$. Note that if

$$\text{Max}(X_i, X_{i+1})F_j = \text{Max}(X_i, X_{i+1})F_{j_{k+1}},$$

it must be at X_i or X_{i+1} for $p > 2$. If this is the case, say at X_i, then the ordering is based on $F_{j_k}(X_{i+1}) > F_{j_{k+1}}(X_{i+1})$. For $p=2$, it is based on

$$F_{j_1}\left(X_1 + \frac{X_2-X_1}{2}\right) > F_{j_2}\left(X_1 + \frac{X_2-X_1}{2}\right).$$

6. Working from smallest to largest values of X_i and largest to smallest values of Y, compute contributions to the total area and first moment using the appropriate algebraic expressions which are based on the geometric property of each entity. Area contributions come only from alternate regions bounded by the X subrange boundaries and ordered entities in pairs. This correctly chooses the inside area contributions. The area and moment equations are derived analytically from the definite integral equations. (This is a tedious but straightforward task.)

$$\text{Area} = \sum_{i=1}^{n-1} \int_{X_i}^{X_{i+1}} \left[F_{j_1} - F_{j_2} + F_{j_3} - \ldots + F_{j_{m-1}} - F_{j_m} \right] dX + \sum_{i=1}^{L} A_{c_i}$$

$$\text{Moment} = \sum_{i=1}^{n-1} \int_{X_i}^{X_{i+1}} X \left[F_{j_1} - F_{j_2} + F_{j_3} - \ldots + F_{j_{m-1}} - F_{j_m} \right] dX + \sum_{i=1}^{L} X_{c_i} A_{c_i}$$

$X_c = \text{Moment}/\text{Area}$

where X_c is the X coordinate of the centroid.

7. Step 3 through Step 6 are repeated interchanging X and Y.
8. The centroid point is displayed contained in a small box, the X, Y values of the point are placed in the X, Y registers and the properly scaled X-calculated area and Y-calculated area are stored in the second set of X, Y registers. "Reject/Accept" is put in the message register.
9. The user may push the accept button, the point is made a dot, the box and message disappear, and the program terminates. If the user presses the reject button instead, the point, box and message disappear, but the contents of the X, Y registers are not changed, and the program terminates.

During steps 1 through 7 the message register contains "Computation Proceeding". If the parameters chosen are limited to point-area centroids instead of closed polyconics, then the program computes the centroid of the system of these centroid points.

CONCLUSIONS

The major limitation of the graphic system is that the software would be highly complex and extensive which, like APT Systems, is costly to produce. Use of standard program-

ming languages and isolation of required basic machine language subroutines will minimize this problem.

The volume of data required to represent graphics in visual form is very high requiring large, cheap storage systems. This is especially true when mathematical capabilities are implemented for processing of higher degree curves and three-dimensional curve and surface perspective projections into two dimensions in a general way. There is expected to be an extensive evolution of console and application program features as graphic systems come into general use. For this reason, modularity in software implementation and large amounts of financial resources will be needed. It is expected that the potential revolutionary effect of graphic systems on design activities will justify such expenditures.

Although this system as partially described here is idealized and not implemented, many of the features will be realized in systems being planned and in development at the Digigraphic Laboratories and other facilities of Control Data Corporation.

It is believed that the goals of this graphic system are met to a great degree by the features described. It is a practical system in that all the features can be implemented within the current technological state. Powerful features are found in the macro capability, the data structure and grouping capabilities, and in the potential power of the application interface. Economy is found in the relatively simple console hardware and in emphasis on minimization of the computer time required. Sophistication is found in the use of order dependent operations, macro availability and potentially in applications which can be easily included in the system. These last mentioned features make the system easy to use on its simplest level, but a challenge to the imaginative and strongly motivated person.

ACKNOWLEDGMENT

The ideas and basis for this system have come from many sources. Besides the authors in the Bibliography, the following people have significantly contributed: Thurber J. Moffett, Jack Antchagno, Joe Koenig, Robert Cushman, Henry Haig, Russel Peterson, James Thebodeau, Chester Small, Paul Downey, John T. Gilmore, Kenneth Gielow, Frank Greatorex, Edward Fitzgerald, Phillip Peterson, Richard Stewart, and Dan Paymer.

REFERENCES

[1]E. L. Jacks, "A Laboratory for the Study of Graphical Man-Machine Communication", *Fall Joint Computer Conference Proceedings,* Oct. 1964.

[2]M. R. David and T. O. Ellis, "The RAND Tablet: a Man-Machine Graphical Communication Device", *Fall Joint Computer Conference Proceedings,* Oct. 1964.

[3]I. E. Sutherland, "Sketchpad, a Man-Machine Communication System", *Spring Joint Computer Conference Proceedings, May* 1963.

[4]Norman H. Taylor, "A Fully Integrated Digitalgraphic Processor", *IRE Professional Journal on Instrumentation,* Page 377, 1962 and private correspondence to Donn B. Parker.

[5]R. Stotz, "Man-Machine Console Facilities for Computer-Aided Design", *Spring Joint Computer Conference Proceedings,* 1963.

[6]B. M. Gurley and C. E. Woodward, "Light-Pen Links Computer to Operator", *Electronics,* Vol. 32, No. 47, November 20, 1959.

[7]D. T. Ross and J. E. Rodriguez, "Theoretical Foundations for the Computer - Aided Design System", *Spring Joint Computer Conference Proceedings,* 1963.

V

APPLICATIONS

Dr. David A. Pope*

On-Line Scientific Applications

CONCEPTS OF ON-LINE COMPUTING

ONE OF THE MOST INTERESTING recent developments in computing is the on-line concept of rapid interaction between the digital computer and the user. This development has immediate application in the area of scientific computing, particularly in those problems which might be characterized as research computations, rather than production computing. In the former type or problem, very often the specific algorithms to be used in the numerical solution are unknown, and a major part of the problem is to find a reasonable set of such algorithms and to demonstrate that they do, indeed, work for the specific problem. In this effort, the possibility of a dialogue between the user and the computer presents some real advantages.

An on-line computing system in this context might be characterized in the following way. Access to the computer should be immediate, at least on the user's time scale. In practice, this means a delay never exceeding a few seconds, and usually less than 0.1 second. The results of a computation should be available immediately, and in a form easy for a human user to comprehend. Finally, the programming should be easily modifiable so that changes in the algorithms can be made quickly, on the basis of intermediate results.

The computational requirements both for the problem being investigated and for the software package necessary to implement the on-line system mean that a digital computer of moderate to large size is involved in the system. This means that, for economic reasons, most on-line computing systems will need to have the large central computer shared by several users; thus some kind of time sharing scheme must be provided which does not conflict with the immediate availability requirement above. Therefore we are led to the concept of a central computer provided with a number of time shared user consoles, each console provided with convenient input-output devices.

*Chief of Programming, UCLA Computing Facility

THE CULLER-FRIED APPROACH TO ON-LINE COMPUTING

One such approach is the on-line system developed by G. J. Culler and B. D. Fried[1]. In this system the user is provided with two typewriter keyboards for input, and a storage CRT for output. The data is presented to the user and computed *functionally*. That is, the basic unit which the user manipulates is a single real or complex valued function, which is represented in the computer by a pair of vectors of up to 125 points each. The output CRT displays a pair of vectors as a set of point pairs (x_j, y_j), $j=1, 2, \ldots 125$, with the adjacent points (x_j, y_j) and (x_{j+1}, y_{j+1}) connected by a straight line. This gives the appearance on the CRT of a smooth curve, which may be interpreted as a real valued function, or as a curve in the complex plane. One typewriter keyboard is used for the designation of storage addresses, each address being given by a number and a letter, such as 1A, 3F, etc. This keyboard is also used for typing alphanumeric information on the display CRT.

The other keyboard is used to designate operations on the functions. On the basic levels, the computer can be operated as a glori-

fied desk calculator. The arithmetic operations add, subtract, multiply and divide are supplemented by the commoner mathematical functions such as square root, sine, cosine, logarithm, exponential, forward difference, and sum. Each operation key, when depressed, initiates a subroutine in the computer which performs the operation on the entire function. To supplement these, there are also operations to load and store functions, and to display one or more functions on the CRT. Thus the user can manipulate functions, displaying the results instantly whenever a display is wanted. As an example of this, we may take the generation and display of the function

$$y = e^{-x^2/2},$$

for $-1 \leq x \leq +1$. The operation keys to be depressed would be: J-generate, square, divide, -2, exponential, store, 1, E, display, 1, A. The J-generate is an operation which generates the standard linear function $y=x$, for $-1 \leq x \leq +1$. This function is then squared, divided by the constant function -2 (entered by numerical keys on the operation keyboard), exponentiated, stored in location 1E, and then displayed on the CRT.

However, this system would be of limited usefulness without some method of conveniently building up more complex algorithms from the simple ones provided. This is done by using the concept of console programming. A special program key is provided which, when depressed, puts the computer in a program writing mode. While it is in this mode, a list is formed of any keys depressed, and this list is assigned to a chosen vacant key. Thereafter, whenever the chosen key is pushed, the computer automatically pushes the entire list of keys which was assigned to it. Furthermore, one key on the list may itself be a programmed key, and call on a sublist of operations. This is done, of course, by automatically providing appropriate linkages between the basic subroutines.

Thus, by nesting subprograms, a single key may be constructed to perform an algorithm of almost any complexity, involving perhaps thousands of operations. In this way, the full power of the digital computer can be utilized, and yet controlled and monitored in detail by the user. This idea also simplifies the modification of programs by the user, since any programmed key, being a subroutine, can be changed without modifying the program of which it is a part, and without influencing programmed keys which went to make it up. In this way, a user can explore his problem, making up and discarding programs, while watching the results of his computation on the CRT. A computing algorithm will thus evolve which will solve his problem, or else perhaps enough will have been learned about his behavior so that it can be reformulated and a fresh attempt made.

EXISTING CULLER-FRIED ON-LINE SYSTEMS

The Culler-Fried system, as described above, was first developed at the TRW Canoga Park facility, and a two station prototype system is now there, using one RW 400 computer for each console, with no time sharing. An advanced model is currently being delivered to TRW/STL in Redondo Beach, which will have four consoles, time sharing one RW 340 computer. Also a Culler-Fried system similar to the prototype system is being developed at the UCLA Computing Facility, which uses the IBM 7094. At present the UCLA system has one console and ties up the 7094 completely, but in a few months it is planned that the Culler-Fried system, along with other on-line systems such as the PAT language will share the 7094 memory together with standby 7094 problems, and the on-line computation will be done on an interrupt basis, but without the necessity of exchanging memory.

KINDS OF PROBLEMS AMENABLE TO THE CULLER-FRIED APPROACH

Various problems have been tried on the prototype system in Canoga Park for approximately two years. The problems for which this computing system is particularly suited may be characterized as "fixed point" problems in a function space. That is, a function f is thought of as an element or "point" in a set of functions, or function space. There is, in this type of problem, an operator T on the space, which maps the function space into itself, and the problem is to find a function f which is "fixed" under T; that is, f satisfies the equation $T[f]=f$. A simple example of this kind of problem is the solution of an ordinary differential equation $y'=f(x, y)$ with initial condition $y(a)=b$. If this is written in integral form, we have

$$y(x)=b+\int_a^x f(t, y(t))\, dt$$

We identify the integral operator T as

$$T[y] = b + \int_a^x f(t, y(t))\, dt$$

and we have the problem expressed in the fixed point form $T[y] = y$. The Picard iteration for the solution can then be written

$$y_{n+1} = T[y_n],$$

yielding successive guesses $y_1, y_2, \ldots$ from an initial guess y_0. Other problems with essentially this same structure are found in partial differential equations, calculus of variations, integral equations, control theory, and other mathematical and physical problem areas.

Given such a formulation of a problem, the problem analyst uses his experience to set up algorithms for solving the fixed point problem, usually with some iterative scheme. It becomes immediately apparent *during the computation* whether or not the algorithm is converging and, if it is not, just where the difficulty lies. This information is then used by the analyst to revise the algorithm and try again. In practice it is found that a great deal of insight into his problem can be obtained by a skilled user.

One of the most significant advantages of this type of on-line system, in fact, seems to be that the human user of the computer does not need to be a computer expert, or indeed to know very much about computers. What he *does* need to know is that area of mathematics and numerical analysis which is involved in his problem. The role of the computer is to do the tedious work, including the tedious programming bookkeeping, and free the problem originator to think about his problem.

REFERENCE

[1]G. J. Culler, B. D. Fried, and D. A. Pope, "The TRW Two-Station, On-Line Scientific Computer", *TRW/STL Physical Research Division Report 8587-6002-RU-000,* Vols. II, III, IV, July 1964.

Dr. R. B. Talmadge*

Structuring Compilers for On-Line Systems

INTRODUCTION

RECENTLY a number of systems† have been produced for on-line operation of a computer job shop which have aimed at improving user productivity by allowing continuous man-machine interaction, while at the same time preserving compatibility with previous systems. Most of these have employed modified versions of compilers written for a batch processing environment. As might be expected, the internal techniques used for handling multiple input strings, especially resource sharing techniques such as multiprogramming and program commutation (time sharing), conflict with the single string orientation of the original implementation. In particular, the program segments themselves, and the intermediate data retained, are much too large; so that system response is degraded because of wasted blocks of core and excessive swap times.

Most of the proposed remedies, such as in-core compilers, and the use of read-only code, while salutary, apply equally well to any program operating within that envoronment. The question naturally arises as to whether distinctly different principles of organization, as contrasted to techniques of mechanization, are desirable. This in turn leads to consideration of the compiler as a system component, to its role in the relation between system and user, and to the question of how much compiler design is influenced by the process one wishes to optimize (in whatever sense that word is being used).

It is the purpose of this paper to suggest that a critical examination of the overall objective of the compilation process leads to an organization founded on the requirements of providing optimal user/system interaction; that this structure is, to a large extent, independent of internal techniques; and that past experience will therefore prove a valuable guide to future development.

USER/SYSTEM INTERACTION

The basic starting point in this exploration is the functional relationship between user and system. This is outlined in **Figure 1**, which illustrates the paths of information flow, and the functions involved. For the system, these functions are *compilation* and *execution;* for the user, *formulation, modification,* and *check-out.* Interaction then occurs in the following way.

1. The user's original formulation of his problem in some source language, or combination of source languages, is entered into the system through the compiler.
2. At some point in the course of the compilation, misuse of the language is detected. Information is returned to the user which causes him to modify the original formulation.
3. When the modified formulation seems satisfactory to both user and system, the prob-

†Three well-known examples are discussed in references[1, 5] and [8].

*Manager, Experimental Systems Group, Los Angeles System R&D Department, IBM.

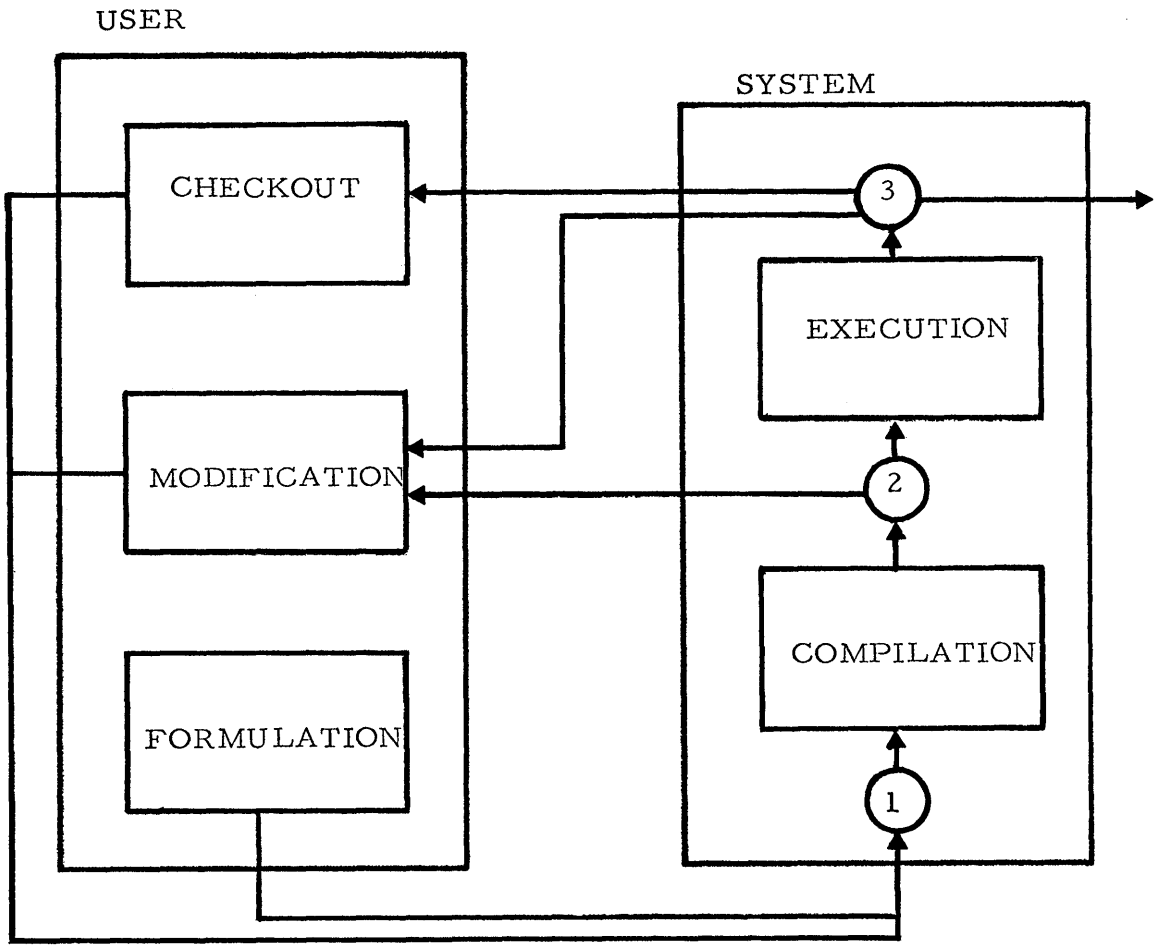


FIGURE 1

USER/SYSTEM INTERACTION

lem proceeds to execution. As a result, information is generated which leads the user to produce further modifications, and statements intended to help check out the program.

4. Eventually (it is hoped) execution is successful, and the program is considered operational.

It is important to realize that this process is essentially the same irrespective of the number and timing of the interactions, and the rate of information flow in the data paths. The picture is not affected, for example, whether or not compilation proceeds to a nominal end before messages are conveyed to the user, or whether execution is interruptable immediately upon the occurrence of some unforeseen condition. Thus, although an optimal system involves some compromise between minimizing the number of iterations of the process, the time per iteration, and the cost of the equipment, there is firm reason to believe that certain structural principles remain invariant to any adjustment.

This discussion, of course, covers only a portion of the actual system. To the user, however, it is the most important part; indeed, it is virtually the only part. For the term compilation, as used here, represents the entire program preparation activity. So that a compiler is a system processor, primarily a language processor, whose function is to turn the user prepared formulation of his problem into a form suitable for use within the hardware-software complex. This is an enlargement, perhaps, of the traditional notion of compiler, but it is pertinent to the course of the discussion. As a complement to this, the term execution is used to represent the carrying out of the intent of the formulation; that is, the actual performance of the stated algorithms.

Thus the interface between compilation and user is fixed, as determined by external considerations; the interface between compilation and execution is variable. As will be seen later, adjustment of this interface can be used to improve the overall operating efficiency. To see how this is possible, to get an idea of the minimum point to which compilation should go and the maximum beyond which it should not proceed, it is necessary to make a closer examination of the user functions (**Figure 1**).

Formulation. Although the mechanics of formulation may be rather complicated, interface with the system reduces to expressing the problem in processing languages. For this discussion interest centers not in the details of a particular language, but in characteristics observable of programming language usage in general. The important point is that the number of problem oriented languages in use today is large, and the number of dialects within these languages is even larger. This trend is not likely to be reversed however much one might wish to the contrary. Linguistic expression is rather personal, so that users tend to specialize in forms which suit their temperament as well as their particular class of problems. Furthermore, even if one were to produce a language capable of expressing all data processing activity, balkanization would occur; not especially out of pertinacity, but because user efficiency is enhanced by languages which permit simple and clear expressions of the problem. Therefore, it is really the responsibility of the compiling portion of the system to handle a variety of languages; and to do so in such a way as to permit easy intercourse between them.

At the same time, it remains a fact that processing requirements are much the same for large classes of problems. Hence, underneath their obvious differences, problem oriented languages fall naturally into families whose members express virtually the same functional capability. An outstanding example of this is that class whose most well known members are FORTRAN, COBOL, and ALGOL.

Modification. There are two ways that modifications to existing programs are done in current systems, depending upon the facilities provided, and the preference of the individual user. First, changes may be expressed in modification units which are independent of the content of the program. A common method, for example, is the ALTER mechanism for replacement, insertion, or deletion in card record files. Second, changes may be expressed in terms of the content of the program; that is, in units depending upon the formal syntax of the source language. For instance, a particular symbol in the program might be replaced, or a string of characters inserted at some arbitrary point. For maximum use and flexibility, the compiler must find a simple way of reconciling these two distinct types of procedure.

Checkout. Checkout embraces all activity required to obtain a correct program statement, from detection of misuse of the language to debugging the logic of the problem. The former is clearly the province of the compiler. As for the latter there are two points of view. Some users regard debugging as a general problem which should be handled in a general way. If this principle is adopted, it leads to the formulation of a separate debugging language, and so to another language responsibility for the compiler. On the other hand, some prefer to use statements in the original source language; and hence to use one of the two modification methods to insert the debugging requests. In either case, the volatile nature of these changes requires that they be inserted at a point which is effectively beyond the permanent information retained by the system.

Most important, however, is that changes, whether permanent modifications or temporary debugging statements, should be on an incremental basis; that is, should not require the entire resubmittal of the program. Reduction in the amount of superfluous data transmitted between elements of the complex is the biggest single factor in obtaining superior performance from both user and system.

FIRST PHASE OF COMPILATION

These considerations, which obviously apply to any system, lead naturally to the expectation that the convergence of function for a related family of languages can be profitably paralleled by a similar convergence in the compilation process; and hence, that the first phase of compilation should be the production of a standard program representation, free of external irregularities, and permitting easy incremental modification. This is, in fact, the

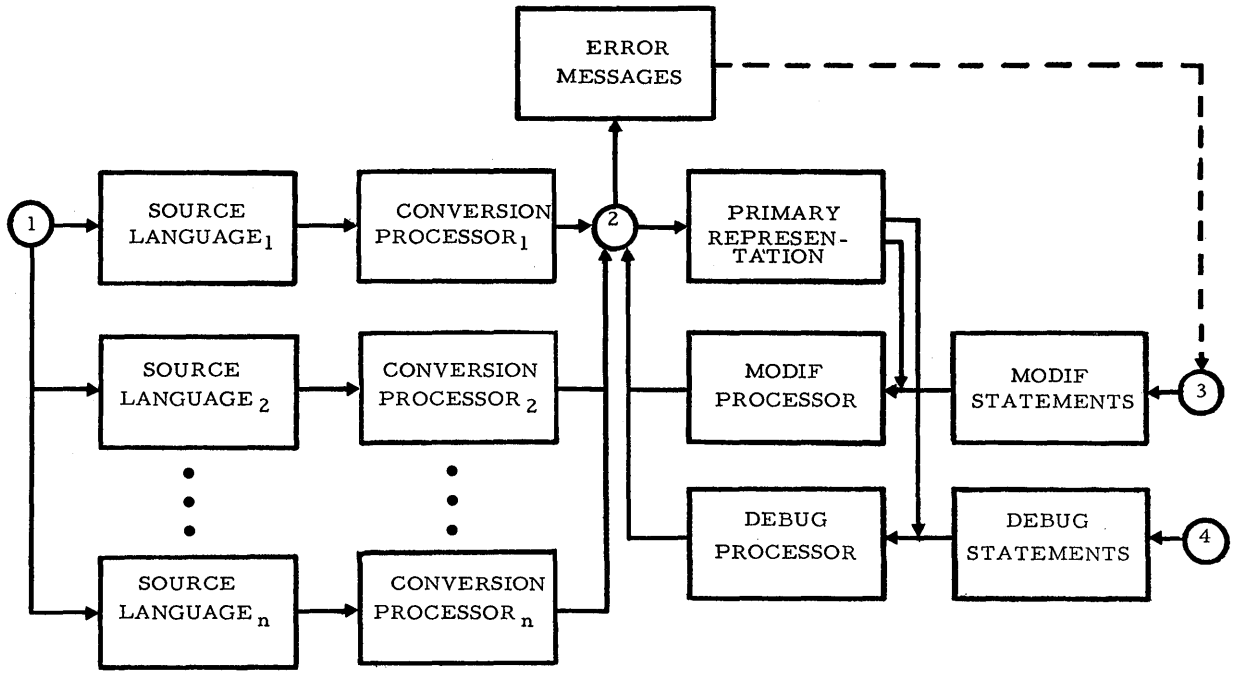


FIGURE 2

FIRST PHASE OF COMPILATION (Family of Languages)

method adopted in most recent compilers.* It results in a functional organization like that illustrated in **Figure 2.**

1. The source languages of the family are treated by individual conversion processors to produce a *primary (internal) representation* of the program.
2. Errors detected by these processors are sent to the user by whatever means are natural for the system. In an off-line system, for example, they would be collected and dispatched as a group. In an on-line system, with the user at a responsive input device, they would be sent immediately, as individual messages leading to possible intervention.
3. In either case, the resulting modifications are handled by a modification processor which replaces individual statements in the primary representation.
4. Similarly, debugging statements which the user generates as a result of execution are passed through the debugging processor.

Of course, the modifications and debugging processors, which are shown in the diagram as separate from the conversion processors because they represent separate functions, would, in practice, be entirely absorbed in them.

As for the conversion processors themselves, their functional capability is delimited by the properties and information desirable in the primary representation. Some of these can be fairly clearly established at this time. First, since the representation is to be free of language errors, conversion must, at least, accomplish a complete lexical and syntactical analysis of the program. Second, as the representation contains all the symbolic information of the original, and serves as the basis for modifications, it must contain explicit markers for modification units and be structured so as to facilitate these changes. This puts an upper limit to the amount of processing which can be done in conversion. For, since even the simplest external change can have profound effects on the meaning of a program, it is ex-

*Such as 7090/94 IBCBC (IBM), 1107 FORTRAN (Computer Sciences Corporation), and QUIKTRAN (IBM). For an excellent discussion of the primary representation used in the last of these, see reference [4].

ceedingly wasteful to attempt any instruction generation, or even to make tentative attempts at optimizing execution efficiency. At the same time, good practice dictates that as much information as will be useful should be collected from the program string as it is converted, and placed in a form most suitable for later processing. Hence the representation will consist of, and the conversion processors must produce, some combination of lists, tables and formal expressions which display the explicit structure of the program (particularly loops and transfers), the usage of symbols, and the characteristics of the data; and facilitate the functions of optimization, instruction generation, and (as will be seen) direct execution.
Of the many advantages of this organization, the following are probably the most noteworthy.

1. Production of the primary representation effectively isolates the external world from the interior of the complex. This makes a substantial part of the system immune to language changes, emendations, and additions. It also permits the introduction of a new language into the family by simply constructing another conversion processor, most of whose pieces are already available.
2. Retention of the primary representation within the system as the permanent symbolic form of the user's program permits additional useful services to be supplied, as well as providing the user with the operational convenience of handling small volumes of data. Special system processors can be easily designed to exploit the precise, explicit information available. A natural one, for example, would be a flow charting program.
3. Data flow is, of course, drastically reduced. Further efficiency is obtained because recompilation starts from an advanced base: an appreciable fraction of the work of compilation is expended in the data gathering and syntactic analysis of conversion. This, in effect, gives a specific meaning to the somewhat loose term incremental compiling (and probably the only sensible one).

There are thus persuasive reasons, dictated by general considerations of user convenience and overall efficiency, for adhering to the structure so far described. It is, therefore, of some interest to turn to the systems functions to see if these reasons are reinforced; and, if so, to determine how much can be deduced about the structure of the rest of the compiler.

DIRECT EXECUTION

With the basic statements of the program resident in the system in internal form, the next step is to consider the boundary separating compilation and execution. The primary factor influencing adjustment of this boundary is the desire to attain a favorable resolution of the inevitable conflict between service to the individual user and service to an entire group of users. Here, for the first time, a distinct difference becomes apparent between an off-line system and a responsive on-line system.

In the latter case, there are a substantial number of programs for which the overall efficiency will be markedly improved by direct (interpretive) execution from the primary internal representation. This arises for the following reasons:

1. There are a number of jobs in any installation which are processed repeatedly by the compiler, and (partially) executed many times solely for the purpose of executing correctly once. Examples abound, but the most obvious ones are student problems at universities, and the desk calculator type so common in the open shop installations of the aerospace industry. On-line systems are, of course, aimed directly at this class of personal computing.* Direct execution can significantly reduce the total effort by bypassing much work that is ordinarily wasted in compiling, re-compiling, and executing incorrect instructions. For in this mode of operation, the immediate availability of debugging information, in fact the absolute necessity of supplying error information which the user could not have anticipated he would need, together with his presence on-line, cuts short the execution of most unwanted sequences.
2. One of the touted advantages of an on-line system is the use of the computer as an intelligence amplifier. In this form of operation the user designs his program as he goes along, presumably building on the results of previous (partial) executions to decide what to do next. Direct execution supplies a convenient, readily available tool

*To use the apt classification of R. L. Patrick[7].

for conversational interaction between program and user. Furthermore, even though such programs are undebugged almost all the time, the interpreter is in full control and so supplies automatic protection for other programs concurrent in the system. Errors which occur result in messages to the individual user without requiring system interruption. The resultant reduction in overhead tends to improve overall efficiency and maintain satisfactory response times.

None of this applies to a batch processing system because the real advantages of the direct method (which still exist) are nullified by the temporal length of the communication paths, and the right granted every program to exclusive use of the machine.

Nevertheless, however well suited direct execution is for some problems, there are others, more numerous in most installations, for which the necessity of repeated high level interpretation is a serious drawback. Even the least experienced user can, and will, write programs for which this would become a problem for him: programs, for example, with loops traversed many times, programs which once checked out are to be used repeatedly; or programs so large as to overstep reasonable limits for interpretation.

These considerations are reflected in current responsive on-line systems, which exhibit a complete dichotomy of attitude. On the one hand, systems which encourage the user to build his programs on a piecemeal basis are dedicated to a particular language and are fully interpretive.* On the other hand, compatible systems provide for interpretation only as a user program, thereby severing any direct connection between this mode of operation and the system processors. The result is an appreciable drop in effectiveness, and hence an overall diminution of system utility. Therefore a system which is to be seriously regarded as general purpose must find a way to integrate both modes easily into a common framework.

Figure 3 illustrates how this can be done. Starting from the primary representation, processing follows one of two paths: either to direct execution, or to compilation in the more traditional sense. The decision as to which path to follow might be left entirely to the user. More likely the total interest would be better served by having system control make the choice subject to general rules laid down by installation management. A simple rule, for example, which would serve many installations well, would be to go to direct execution with (pieces of) programs which are yet undebugged. More sophisticated rules depend upon how much the installation is willing to acknowledge direct execution as an important option in any well designed on-line system.

An interesting observation on this point is that since many currently announced computers use read only memory for control and instruction interpretation, a little care in the design of the primary representation would make it possible to do most, perhaps all, of the interpretation in the hardware. This is tantamount to direct execution of a symbolic program (hence the choice of a name), a subject of some interest today[6]). The gain in efficiency would greatly enlarge the class of programs for which direct execution is a distinct advantage.

SECOND PHASE OF COMPILATION

If the choice is to proceed to hardware execution, the second phase of compilation is entered. The object of this phase is to convert the primary representation to *program text* which is the interface to system execution control, and hence to execution proper. Production of this text, the analogue of the relocatable code used in most current systems, is carried out in the following manner, by a process which separates naturally into the functions of *optimization, instruction generation,* and *assembly* **(see Figure 3).**

Optimization. The first step is to apply an optimization procedure to the primary representation, in order to improve the execution performance of the program. This involves such activities as flow analysis; noting where auxiliary calculations may be better performed, and where they may be suppressed; analysis of loop structure to determine positional indicator usage, common expression pre-calculation, and the possibility of loop collapse; all of which are done now in most compilers, though not, perhaps, at quite the same place. That this is the proper place for

*Of these, JOSS[9] and QUIKTRAN[3] are perhaps best known.

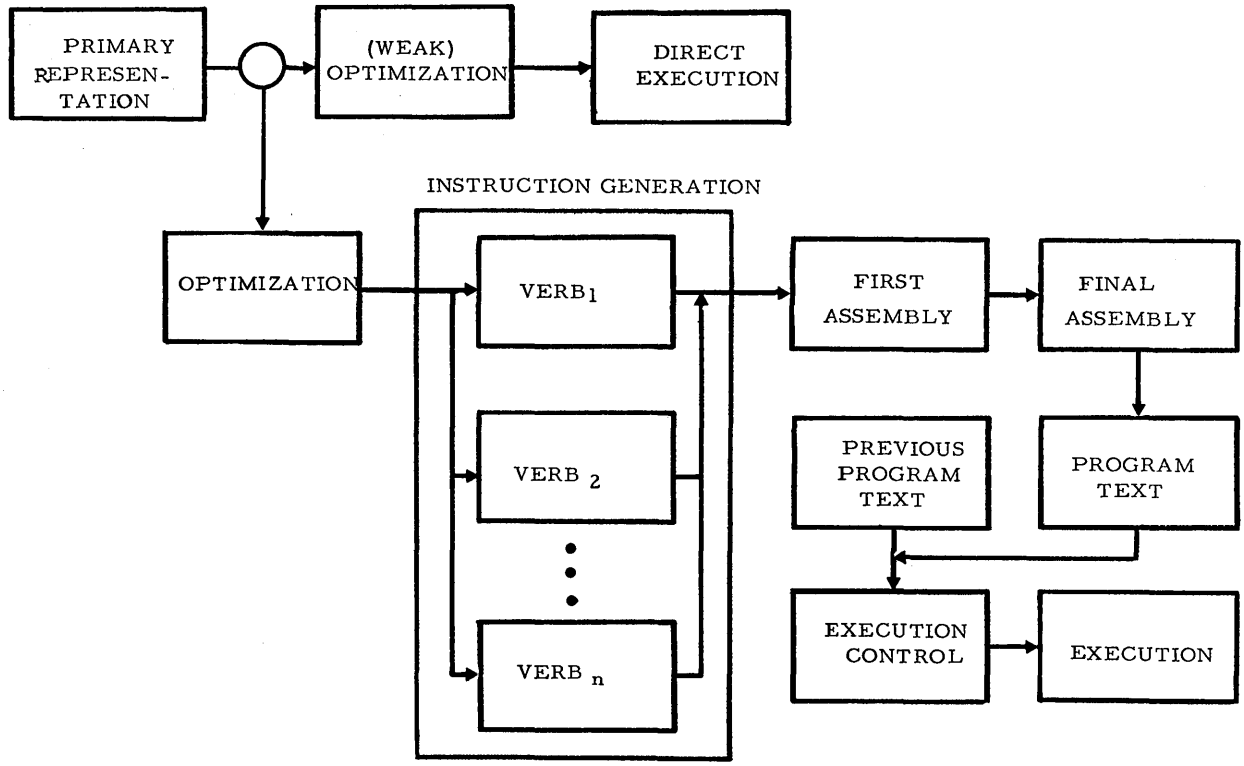


FIGURE 3

SECOND PHASE OF COMPILATION

such action is not hard to justify. It must occur following all modifications, since it is meaningless to optimize without knowledge of the entire context of the program; and it should precede any instruction generation, since preventive measures are always better than remedial ones.

There is a further advantage in that it permits the easy mechanization of considerable latitude in the amount and type of optimization applied to any given program. A user, for example, might choose to skip all, or part, of the procedure if he has good reason to believe it will not notably improve his program. In this connection, the box marked "(weak) optimization" in **Figure 3** merely signifies that some type of gross optimization, for example the loop collapsing analysis, might well be useful prior to direct execution.

It should also be noticed that this step completes the catalogue of information desirable in the primary representation, and so serves in implementation for the final determination of the functions undertaken by the conversion processors.

Instruction Generation. The function of instruction generators is to interpret statements within the (optimized) context of the program, and the hardware to be used for execution, to decide which instructions are to be used. At this stage, the verbs of most procedural languages can be handled by autonomous processors. Within these processors most remaining decisions can be made by selecting one of several pre-planned possibilities, according to the descriptions of operands within the scope of individual operators. For example, if the expression A+B is to be calculated, the addition generator would examine the data descriptions of A and B: if they were floating point numbers, a simple floating addition sequence would be used; if fixed point, some preliminary scaling might be required, as well as the use of fixed point hardware operations. Thus the organization favors a high degree of modularity in the compiler; it also localizes treatment of most changes in the interpretation of a statement to an easily accessible place.

It must be clearly understood that the gen-

erators, in spite of their name, are confined to making decisions about what instructions to use, but do not themselves act to issue the instructions. This takes on added significance in view of the fact that these decisions are the same as those made in the implementation of direct execution, for it means that the same routines can serve this function in both paths of the compilation process.

First Assembly is the name given to a small set of routines which operate concurrently with instruction generation to implement the decisions made by the generators. Hence, the main functions undertaken by these routines are to substitute particular instances of operands into lists of instruction forms, to (logically) separate and count the independent streams of instructions, and to record usages of symbols which will result in interprogram references. In this effort considerable use is made of advanced assembly techniques such as multiple location counters and deferred symbol definition. First assembly, however, should not be confused with the first pass of a typical assembler, most of whose work is expended in conversion.

In the direct mode this function is replaced by execution. More precisely, first assembly is supplanted by a routine which carries out, or attempts to carry out, the intent of the instructions produced after the operand substitutions are made.

Final Assembly embraces preparation of the form required by execution control; and, perhaps, production of supplementary information for the user (such as a listing). Little can be said about the implementation of either of these functions since the processing is very much dependent on the form adopted for the program text, and the sophistication of the techniques used to take advantage of the available hardware. It should be noted, though, that the supplementary information, so common in off-line systems, is of no practical utility to an on-line user and might well be eliminated.

RELATION TO ON-LINE SYSTEMS

The organization thus described embraces all the functions assigned to the compilation process, so that the picture of compiler structure is essentially complete. Furthermore, the discussion has emphasized that the considerations used in developing this picture apply to most systems, being based on overall optimization of the user/system relation. But in actual practice the compiler must operate, and produce code which operates, consistently well within the internal conventions and techniques of a particular environment. Therefore, to verify that the description is sound, that the structure is not just suitable but desirable, it is necessary to consider requirements which are peculiar to dynamic environments, particularly on-line systems.

The salient feature of such systems is that they attempt to amplify the effective computing power of the hardware by the use of techniques which permit multiple concurrent users. Multiprogramming and multiprocessing, for example, exploit the possibilities of parallelism, while time sharing exploits the disparity in human and computer speeds. Much of the advantage gained by these techniques would be dissipated without effective resource allocation and minimization of system overhead. Segmentation of programs into fairly small pieces is by far the most significant factor in this effort, on both counts. First, it improves core utilization by reducing the occurrence of unused blocks. Second, it reduces system overhead because the presence of pieces of many programs in core at the same time has a double effect in diminishing the number of words swapped between core and backup storage.

Now, it has already been observed that the organization presented is favorable to fragmentation of the compiler itself. Moreover, the functional division simplifies use of techniques for dynamic reduction of the operating size. For example, the independence of individual instruction generators permits implementation in which only those generators actually in use by any program need be brought into core, where they can remain until not needed. Further improvement can be obtained by designing the primary representation to separate the global information from the local, which reduces the amount of data that has to be swapped in compilation; and by limiting the size of any segment which is compiled down to program text, thus limiting the space needed for such data.

Limitation of the size of segments is a technique practiced in many current systems. Its success depends upon having an execution control which can readily build, and efficiently

operate, a program composed of smaller segments. In attempting to do this for on-line systems, there is much to be learned from software solutions already in use. That subject, however, would need a lengthy discourse to do it justice. In this discussion, it is possible only to touch briefly upon a few topics of general interest.

First of all, program segmentation has its darker side. It increases the core management problem because of the appearance of fragments: small pieces left over in allocation which must somehow be collected into useful sizes. Similarly, program protection is more difficult, for occasions arise when non-contiguous segments of the same program must reside in core simultaneously. The difficulties of interprogram communication are also magnified: the smaller the pieces, the more likely references will be external to a given segment. The first two of these problems occur only in dynamic systems, while the last has been around for some time. In attacking it, current systems have developed program text and loading techniques which could be quite useful if properly converted to a dynamic environment.

In those systems, program text consists of instruction strings in which the addresses are supplemented by relocation bits whose normal function is to indicate whether the address is constant or relative. For interprogram references, however, the encoding refers to a dictionary in which enough symbolic information is retained to identify the desired reference. It is the function of that portion of execution control called the loader to combine various segments into a single operating program, as desired by the user. In this process, the text is translated to specific core locations by interpretation of the relocation bits in conjunction with the combined dictionaries. In addition, for programs too large to fit into core, the user indicates how the program is to be prepared for retrieval of specified parts, (called overlays) by execution control.

Now in on-line systems this pre-planned retrieval and pre-calculated translation is better done dynamically. Not only would this improve efficiency by fetching only those portions of a program actually needed during a particular execution, but also it would provide relief from the fragmentation problem by severing the ties to specific locations. But implementation of such dynamic control entirely by software involves considerable overhead during swaps, and in execution, so that some hardware assistance is required. Two approaches are currently in vogue. First, there are page schemes in which segmentation and retrieval are tied to core blocks of hardware determined size and location. Second, there are address translation schemes in which the illusion of a contiguous program is obtained by comparing all effective addresses generated during execution with a translation dictionary to obtain a true address. Again the page concept is in evidence, since the translation shifts the low order bits of the address.

Neither of these methods really eliminates much pre-planned effort on the part of the user or the system. Therefore it is surprising that there have been no serious attempts to embed the loading function into the hardware; that is, to design the CPU to execute program text directly, relocation bits, dictionary, and all. Such an approach, which requires no more hardware than others proposed, has several distinct advantages.

1. It is quite conservative of space. The relocation bits can be absorbed into the address without any practical limitation on segment size. Furthermore, the dictionary is only as large as required by the given program.
2. It is efficient in execution. Address comparison occurs only for instructions which specifically request it.
3. Segmentation is performed according to the natural division of the user. The elimination of artificial fences in core relieves system processors, for one, of the problem of adjusting recalcitrant segments to prespecified block sizes[2].
4. Apart from the dictionary, the programs are absolutely location independent. If read only code is used, very little additional software is needed to implement a scheme in which segments of operating programs are swapped in, but need never be swapped out.
5. Flags in the dictionary can be used to provide program protection on an individual basis, not tied to any particular core locations, and for any number of independent programs. Similarly, dictionary flags can

be used to trigger a segment retrieval mechanism.

For these reasons, design of a program text in conjunction with hardware merits serious consideration. In this design, it is, of course, desirable to make use of negative as well as positive information gained from previous software experience. For example, the program text of several current systems, contains a dictionary for debugging as well as a dictionary for interprogram reference. Because of the reduction in symbolic content, and because the nature of undebugged programs is such that one cannot anticipate what information will be needed, or when it should be obtained, considerable skill is required of the user to keep this dictionary of manageable size and still have it fulfill its purpose. Moreover, loading is complicated by having to undertake the additional functions of interpretation and insertion of debugging requests into the program prior to execution. It is operationally superior in all respects to eliminate the debugging dictionary and rely upon insertion of requests into the primary representation.

Objection might be raised that this is not suitable for code produced by assemblers. Such, however, is not the case. With the previous discussion as background, it is not difficult to quickly arrive at the functional organization of a processor for an assembly language family. **Figure 4,** for example, shows the plan of a macro assembler which would take source language through conversion and first assembly to a primary representation, and from there through final assembly to program text identical in form to that produced

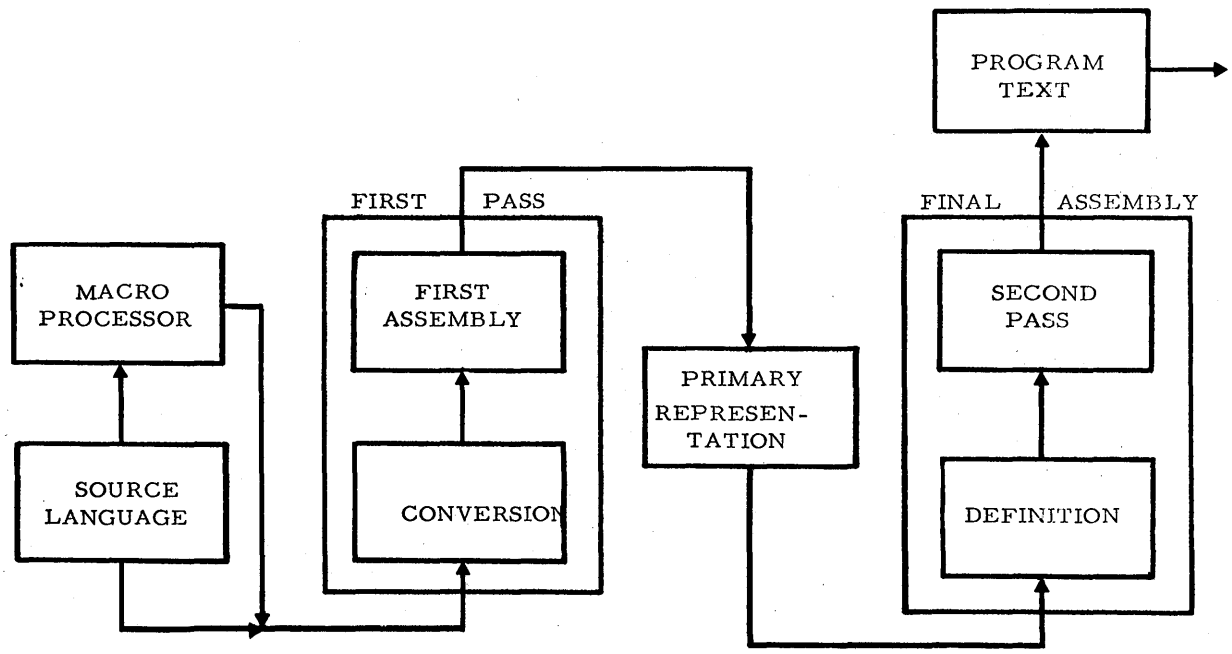


FIGURE 4

ASSEMBLY LANGUAGE PROCESSOR

by any other system language processor. The primary representation, which is naturally quite different from that of a problem oriented language family, again serves as the resident symbolic form of the program, and as the point for incremental modifications. A significant feature of this structure is that instruction generation and first assembly occur prior to output of the preliminary representation. Hence the processing time from it to program text is substantially less than that of other families. Thus, user convenience, uniformity of procedure, and overall operating efficiency combine to form a powerful inducement for a return to completely symbolic debugging.

SUMMARY

But to pursue these ideas farther would necessitate a depth of detail far beyond the scope of this paper. So, in conclusion, it is perhaps in order to summarize the main points of this discussion. First, consideration of the basic relationship between user and system, of the existence of many languages and the need for incremental changes, leads to a compiler whose first phase is dedicated to the production of a primary, internal, representation intended for residence in the system. This step is independent of any internal considerations. Second, because of rapid intercommunications, on-line systems are particularly suitable for direct, interpretative, program execution. The primary representation then provides a means of unifying two distinct modes, interpretation and conventional compilation, at the same operational level. Third, a functional organization for the second phase of compilation based on current practice fits nicely with the special requirements of the internal dynamics of on-line systems. Finally, past experience indicates how compiler output might be designed, in conjunction with hardware, to alleviate some of the problems of segmentation, core allocation, and program protection.

REFERENCES

[1] F. J. Corbato, et al., *The Compatible Time Sharing System — A Programmer's Guide,* The MIT Press, May 1963.

[2] F. H. Dearnley and G. B. Newell, "Automatic Segmentation of Programs for a Two-level Store Computer", *The Computer Journal,* Vol. 7, No. 3. Oct. 1964 (185-187).

[3] T. M. Dunn and J. H. Morrissey, "Remote Computing —An Experimental System, Part 1: External Specifications", *Proceedings of the Spring Joint Computer Conference,* 1964 (413-423).

[4] J. M. Keller, E. C. Strum, and G. H. Yang, "Remote Computing—An Experimental System, Part 2: Internal Design", *Proceedings of the Spring Joint Computer Conference,* 1964 (425-443).

[5] H. A. Kinslow, "The Time-Sharing Monitor System", *Proceedings of the Fall Joint Computer Conference,* 1964 (443-454).

[6] J. E. Meggitt, "A Character Computer for High-Level Language Interpretation", *IBM Systems Journal* 3, No. 1, 1964 (68-78).

[7] R. L. Patrick, "Measuring Performance", *Datamation,* July 1964 (24-27).

[8] J. I. Schwartz, E. G. Coffman, and Clark Weissman, "A General Purpose Time-Sharing System", *Proceedings of the Spring Joint Computer Conference,* 1964 (397-411).

[9] J. C. Shaw, "JOSS: A Designer's View of an Experimental On-Line Computing System", *Proceedings of the Fall Joint Computer Conference,* 1964 (455-464).

This paper reflects the author's personal experience and beliefs. It does not necessarily reflect current product compiler development by the IBM Corporation.

John H. Morrissey*

The Quiktran System

INTRODUCTION

SYSTEMS involving access to a central computer by geographically distant users have been employed experimentally since the very earliest application of computers themselves[1]. During the early 1950's, these systems were commonly oriented towards the substitution of data transmission for the physical transportation of the user or his data between remote locations and a central computer site. This enabled some people to gain computer access and provided others with increased convenience and improved job turn-around times.

During the second half of the last decade, military command and control systems (e.g. SAGE) involving concurrent access to a large computer from numerous consoles, provided a first indication of the potential advantage of coupling man and machine in processing complex problems.

These early military systems stimulated the commercial application of the man-machine concept in the airline, brokerage, and banking industries. These systems are often characterized by special purpose equipment and tailored software which are oriented towards a specific commercial application with emphasis on the maintenance and retrieval of data files.

At the present time such customized teleprocessing systems are being generalized as general purpose equipment and comprehensive on-line operating systems which are oriented towards a wide spectrum of scientific applications with emphasis on both the processing of programs and manipulation of data files.

*IBM System Research and Development Department

SYSTEM CONCEPTS

There are several feasible system approaches to provide remote scientists and engineers with many of the computing services long available to users located close to a digital computer installation.

Batch Processing

One is to envision the remote terminals as merely another type of I/O device. The simplest form of this approach is to limit these terminals to operating system input/output (e.g. SYSIN and SYSOUT) and exclude their assignment to user problem programs. Programs, data, and associated control information are entered and stored awaiting computer availability. This can be scheduled in several ways: the simplest is to await completion of the current job, and then intersperse the stacked remote job into the input stream. Following conventional processing, results are stored awaiting later transmission to the remote output unit. If no other remote jobs are ready for processing, the operating system selects the next job from its regular input source. Concurrently, results are transmitted (under control of either the main processor itself or else an associated I/O processor) back to the remote sites.

This remote batch approach has several advantages

1. It requires a minimum of extra equipment and little additional software development.
2. It is a logical extension of conventional batch processing procedures.
3. Processing efficiency is high.
4. Turnaround time is improved.

5. It is particularly suitable for "express jobs" (e.g. those involving small I/O volumes, no operator set up, and limited execution times.).

But it has the following limitations:

1. Output print volume often (especially during program testing) overwhelms the capacity of the communications terminal equipment.
2. Although turnaround is improved, there is no opportunity for direct, sustained man-machine communication.

Shared Processing

A second, and currently more popular, approach is to envision the remote terminals as computer consoles. This implies that the remote user should have immediate and sustained control over the central computer. Since it is economically unfeasible to have one user dominate the computer, its facilities (time and storage) are dynamically shared among a number of users. This form of multiprogramming is often called time sharing, although another term such as "facility sharing" or "resource sharing" would be more appropriate since much more than the sharing of computer time is involved.

SYSTEM REQUIREMENTS

In the design of any computer system the following major considerations must be analyzed and evaluated: Who is the user, what functions are needed, at what cost, and for what purpose? More specifically, what size computer, how many users, what functional capability, and at what cost per user?

The first factor can be recast into a question of extending an established market by enabling conventional computer applications to be performed in a new or improved manner and creating a new market by providing new capabilities not previously available.

Capability must be structured in terms of price and function. In 1960-1961, when the first generation of time sharing systems were being designed, analysis of existing computers indicated the following:

1. A comprehensive service could be made available to 20-30 terminals time sharing a large-scale computer. Quantitatively this led to the approximation of $4,000 per user per month; ($100K system)/(25 users).
2. A limited service could be made available to 30-50 terminals time sharing a medium scale computer. Quantitatively this led to the approximation of $1,000 per user per month; ($40K system)/(40 users).

$4,000/month computers were widely available and undergoing rapid improvement; $1,000/month computers were not commonly available. Two markets were considered: install computers on large customer premises for shared usage by his employees, and service small users by leasing time from central IBM Data Center installations. The basic system goal was to realize new marketing and technical innovations; consequently, it was the medium scale system with limited capability approach that was selected.

The next consideration was to determine what functions should be provided. This can be subdivided into two areas: qualitatively, what does the market need; and quantitatively, what performance is possible.

The qualitative factor can be recast into an evaluation of the relative merits of a commercial or of a scientific orientation. The following factors were analyzed:

1. Amount of program development vs. production work
2. Importance of job turnaround vs. computer throughput
3. Amount of I/O vs. computing
4. Amount of retrieval vs. computing
5. Programming languages in use
6. Conversion and transition problems
7. Amount of application support required
8. Random storage requirements (space, speed, and price)
9. Density of potential users
10. Reliability requirements
11. System load-up rate
12. Sales expense
13. Customer training requirements.

It was concluded that, although the commercial market was undoubtedly much larger, there was substantially greater immediate revenue potential and lower technological exposure associated with a scientific oriented system.

The quantitative factor can be recast into how computer performance is considered. First, the computer engineer is concerned with throughput, that is, the raw processing power of the system (often expressed in micro-

second units). Next, the computer center manager is concerned with turnaround time; that is, the interval between submission of a job and the return of output (often expressed in hourly units). Finally, the computer user is concerned with solution time; that is, the time interval between the decision to utilize a computer and the receipt of correct results (often expressed in weekly or monthly units).

It was decided to place primary emphasis on the last factor since it is of paramount importance to the new user and is a main source of dissatisfaction with many current users.

SYSTEM DESIGN

Objectives

These system requirements were then translated into the following design objectives. First the price, a $40,000 per month central computer configuration would be shared by 40 remote terminals. Next, performance; speed would be extrapolated downward from existing small scientific computers. Relative to this class of computer: response rate would be under 10 seconds; execution speed would be in the range of 0.1-1; and compilation speed would be in the range of 10-100. Space would be equivalent to that available in small computers; that is 3,000-4,000 words. Finally, language; a source language would be consistently used for program statement, problem debugging, and terminal operation.

Approach

It was immediately realized that the computer price objective would be exceeded if special device design or equipment modifications were undertaken. Hence, a design constraint to use only standard computer products was imposed.

It was also soon apparent that the performance objective could not be realized merely by tailoring existing compilers for operation in a time shared environment because they:

1. Were too large for either in-core residence or for segmented swapping
2. Strongly emphasized object performance at the expense of compiling speed
3. Were not well designed for efficient recompilation
4. Were not effective for source language debugging
5. Generated far too much output data for terminal operation.

It was concluded that a new program design was necessary. Two influences were paramount in deciding on the approach. First, previous experience in using the 701 Speedcode[2], 705 Print[3], 705 Sale[4], and G-15 Intercom[5] systems had demonstrated the power of interpretive execution in the formulation and debugging of small to medium scale scientific programs.

Second, previous exposure to list processing concepts in IPL-V[6] had indicated that list

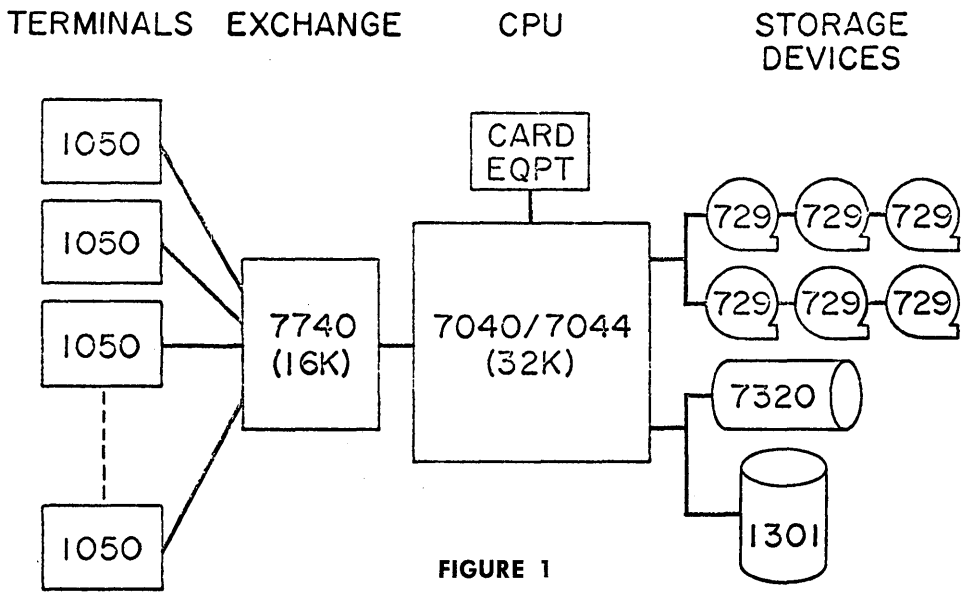


FIGURE 1
EQUIPMENT CONFIGURATION

techniques should be particularly applicable to the design of incremental language translators.

It was relatively simple to settle on the best approach to meet the language objective. Because of the importance of language compatibility, extent of usage, and ease of learning, it was decided to base the language on a subset of FORTRAN equivalent to that available on small computers.

EXTERNAL SPECIFICATIONS

Since detailed information on QUIKTRAN's external design is available[7], only a brief outline is presented.

Equipment Configuration (Figure 1)

1050 terminal for user console
7740 exchange for communications control
7040 computer for all processing
7320 drum for program swapping
1301 disk for user program libraries

Language Specifications

1. Program statements, a FORTRAN subset, are used to write programs.
2. Console statements are used to load programs, to start and stop execution, and to enter and display data.
3. Alteration statements are used to insert, change, and delete program statements.
4. Display statements are used to display source program listings and storage dumps, pre-execution cross reference listings, in-execution data and flow tracing; and post-execution history of data usage and control flow.

Operating Modes

1. Batch: for the remote entry of a complete job for conventional execution.
2. Command: for use of the remote terminal as a symbolic desk calculator.
3. Program: for the conversational entry and execution of FORTRAN programs.

Operation

Use of the system can be best illustrated by seeing it in actual operation.*

The terminals shown in **Figure 2** are connected on a typical operation.

Equipment components are shown in **Figure 1**.

*A 20-minute motion picture was shown at the Symposium.

QUICKTRAN*

Line No.	User	Location	Distance
1	University	San Francisco, Calif.	3,000
2	Aerospace	" " "	"
3.	"	Los Angeles, Calif.	"
4	"	" " "	"
5	"	" " "	"
6	Chemical	Charlestown, W.V.	500
7	Petroleum	New York, N. Y.	1
8	Education	" " "	1
9	IBM Engineering	Poughkeepsie, N. Y.	75
10	" "	Endicott, N. Y.	150
11	IBM Systems Res.	San Francisco, Calif.	3,000
12	" " "	Boston, Mass.	150
13	" " "	New York City	1
14	" " "	" " "	1
15	Demonstration (Movie)	" " "	0

*Users on system when filming movie (9/24/64).

FIGURE 2

TERMINAL LOCATIONS

```
                    COMMAND
101. -READY      N=1+1
101. =           N=           2
101. -READY      Z=1.+10.+100.
101. =           Z= 0.11099999E 03
101. -READY      EDIT(F15.8)
101. -READY      Z=1.-2.*3./4.**5.
101. =           Z=     0.99414063
101. -READY      Z=1.-2.*(3./4.)**5
101. =           Z=     0.52539063
101. -READY      Z=1.2345678*8.7654321
101. =           Z=    10.82151997
101. -READY      Z=1.234E01*4.321E-02
101. =           Z=     0.53321139
101. -READY      Z=SQRT(1024.)
101. =           Z=    32.00000000
101. -READY      Z=LOGF(SIN(1.23)**2+COSF(1.234)**2)
101. =           Z=    -0.00251082
101. -READY
```

FIGURE 3

SYMBOLIC CALCULATOR

When the terminal is in use as a symbolic desk calculator **(Figure 3)** each user action elicits a computer response and vice versa. Each entry is used to introduce a new concept: integer values, floating point values, output formatting, arithmetic operation: +, −, *, /, **, use of parenthesis, high precision, scaling, function evaluation, and expression evaluation.

```
                    PROGRAM ROOT
102. +READY      1 Z=0.5*(Z+X/Z)
103. +READY          GO TO 1
104. +READY CC       X=25
104. =              X= 0.25000000E 02
105. +READY CC       Z=X
105. =              Z= 0.25000000E 02
106. +READY          START 0
                    HALT
103. =BREAK  EXECUTION INTERRUPTED BY TERMINAL USER-LAST
             EXECUTED  102.
107. +READY CC       Z=Z
107. =              Z= 0.49999999E 01
108. +READY
```

FIGURE 4

CONSOLE OPERATIONS

In the program mode **(Figure 4)** (e.g., entry of a program) some of the console commands are start, stop, entry, and display. The user indicates his intention to enter a program (by typing PROGRAM) and gives it a name (e.g. ROOT) so that he can later retrieve it from his library. The program consists of only two statements: the first to evaluate Newton's (or Heron's) formula for the square root; the next to set up an infinite loop. The user indicates the number whose root he desires (X=25) and initializes the iteration variable (Z). Then he begins execution (via START 0), lets it run a few seconds, and then stops execution (via HALT). The system indicates the location of the last statement executed (analogous to a console's instruction counter lights). The user then displays the current value of the square root and it is seen to be very close to the exact answer.

Entering a program to solve the differential equation $dy/dx=X*Y$, $X_0=0$, $Y_0=1$, is an example **(Figure 5)** which illustrates many of the common clerical and syntactical errors usually committed in writing a FORTRAN program. The system immediately responds with a diagnostic message and the user can then easily correct the error. Such mistakes although trivial, often use up several machine runs, each involving several hours delay in advancing towards a checked out program. Notice also that the user may execute sections of the program as they are entered and verify results as he proceeds.

```
                    PROGRAM DIFEQ1
102. +READY         DELX=0.2
103. +READY      1 X=0
104. +READY         Y=.)  -
104. +ERROR  INPUT CANCELLED
104. +READY         Y=1
105. +READY PRINT 2
107. +READY         PRINT 1
107. +ERROR  STATEMENT 1 PREVIOUSLY DEFINED OR REFERRED
             TO AS EXECUTABLE
108. +READY         PRINT 2
109. +READY         FORMAT(5X)
109. +ERROR  THIS STATEMENT MUST BE NUMBERED
110. +READY      2 FORMAT(5X,1HX,5X,1HY)
111. +READY         PRINT 4,X,Y
112. +READY      4 FORMAT(F7.2,F8.5)
113. +READY         START 0
108. =O 02         X      Y
111. =O 04      0.    1.00000
112. =CYCLE  END OF PROGRAM ENCOUNTERED DURING EXECUTION
114. +READY         DELY=X*Y*DELY
115. +READY         Y=Y+DELY
116. +READY         IF(X-1)3,3,5
116. +ERROR  MIXED MODE
117. +READY         IF(X-1. 3,3,5
117. +ERROR  PARENTHESES NOT IN BALANCE
118. +READY         IF(X-1.)3,3,5
119. +READY         PAUSE 1
119. +ERROR  STATEMENT AFTER IF, GO TO, STOP, OR RETURN
             MUST BE NUMBERED
120. +READY      5 PAUSE 1
121. +READY         GO TO 2
121. +ERROR  STATEMENT 2 PREVIOUSLY DEFINED
             OR REFERRED TO AS FORMAT
122. +READY         GO TO 1
123. +READY         END
```

FIGURE 5

PROGRAM ENTRY

Figure 6 illustrates program testing. The user has pursued his completed program. He initiates execution (via START 0) and an alphabetical heading and a line of numerical values print correctly. Then the system indicates that at line 114 an attempt has been made to use a variable before it has received a value. The user detects his error, deletes line 114 and replaces it with the corrected formula. He then indicates the end of alteration (so that the system can run a check to insure that he did not introduce new errors while making the change) and starts execution again.

```
                START 0
108. =O 02       X      Y
111. =O 04      0.    1.00000
114. =ERROR  VALUE OF VARIABLE MAY NOT BE USED
              UNTIL IT HAS BEEN SET
125. +READY         ALTER 114./114.
114. +ALTER         DELY=X*Y*DELX
114.1+ALTER         ALTER*
126. +READY         START 0
108. =O 02       X      Y
111. =O 04      0.    1.00000
118. =ERROR  TRANSFER POINT N DOES NOT EXIST
127. +READY         ALTER 111./111.
111. +ALTER      3 PRINT 4,X,Y
111.1+ALTER         ALTER*
128. +READY         START 0
108. =O 02       X      Y
111. =O 04      0.    1.00000
111. =O 04      0.    1.00000
111. =O 04      0.    1.00000




111. =O 04      0.    1.00000
111. =O 04      0.    1.00000
                HALT
112. =BREAK  EXECUTION INTERRUPTED BY TERMINAL USER-LAST
              EXECUTED  111.
```

FIGURE 6

PROGRAM TESTING—1

Another error shows up at line 118 where an attempt is made to transfer to a non-existent statement (note: in later versions of the system this error is detected before execution). The user corrects this error by numbering the statement at line 111 with a label of 3. He starts execution again. Everything looks fine except that nothing is happening! The user performs a HALT, quickly begins to make a correction **(Figure 7)**, cancels it, and then inserts, after line 113, a statement to increment the independent variable X. Once again, he starts execution and (finally) the program runs to completion (P1 at line 120).

Being curious about the effect of step size on precision, the user then changes the step (DELX=0.25), and executes again (being

```
129. +READY         ALTER 113./113.XXXX-
129. +ERROR  INPUT CANCELLED
129. +READY         ALTER 113.
113.1+ALTER         X=X+DELX
113.2+ALTER         ALTER*
130. +READY         START 0
108. =O 02       X      Y
111. =O 04      0.    1.00000
111. =O 04      0.20  1.04000
111. =O 04      0.40  1.12320
111. =O 04      0.60  1.25798
111. =O 04      0.80  1.45926
111. =O 04      1.00  1.75111
120. =P 1
131. +READY CC      DELX=0.25
131. =             DELX= 0.25000000E-00
132. +READY         START 1
108. =O 02       X      Y
111. =O 04      0.    1.00000
111. =O 04      0.25  1.06250

111. =O 04      0.50  1.19531
111. =O 04      0.75  1.41943
111. =O 04      1.00  1.77429
120. =P 1
133. +READY
```

FIGURE 7

PROGRAM TESTING—2

careful not to execute the program statement that initializes DELX). The program runs to completion and he notes that an increase in step size (DELX=0.25 instead of 0.20) caused a difference (Y=1.77429 instead of 1.75111) in the value of Y. This illustrates some of the potential of close man-machine interaction in the area of numerical analysis.

The next example **(Figure 8)** illustrates use of some of the source language debugging features. First, there is a DUMP of memory

```
                DUMP
     =          DELX= 0.25000000E-00
     =          DELY= 0.55446625E 00
     =          X= 0.12499999E 01

     =          Y= 0.23287582E 01
134. +READY      EDIT(F15.8)
135. +READY      DUMP
     =          DELX=     0.25000000
     =          DELY=     0.55446625
     =          X=      1.25000000
     =          Y=      2.32875824
136. +READY      INDEX
      INDEX      1      +103.  -122.
      INDEX      2      +110.  -108.
      INDEX      3      +111.  -118.
      INDEX      4      +112.  -111.
      INDEX      5      +120.  -118.
      INDEX  DELX       +102.  -113.1 -114.
      INDEX  DELY       +114.  -115.
      INDEX *DIFEQ1
      INDEX  X          +103.  -111.  +113.1 -114.  -118.
      INDEX  Y          +104.  -111.  -114.  +115.
137. +READY      INDEX(X)
      INDEX  X          +103.  -111.  +113.1 -114.  -118.
138. +READY      CHECK
      CHECK *DIFEQ1
139. +READY      AUDIT
     AUDIT        122. /123.    NOT XEQ
     AUDIT       DIFEQ1 NOT  SET
```

FIGURE 8

DISPLAY STATEMENTS

under control of two different output formats. Next there is an INDEX cross reference listing of control flow and data usage. Finally, there is a CHECK for potential errors, followed by an AUDIT, a post-execution history of actual control flow and data usage.

In the next example **(Figure 9)** the user performs a LIST of the program he just finished testing. A variation of this would enable him to punch a card deck ready for processing by any other FORTRAN compiler.

```
                LIST
101. =       CF    PROGRAM DIFEQ1
102. =             DELX=0.2
103. =           1 X=0
104. =             Y=1
105. =       PRINT 2
108. =             PRINT 2
110. =           2 FORMAT(5X,1HX,5X,1HY)
111. =           3 PRINT 4,X,Y
112. =           4 FORMAT(F7.2,F8.5)
113.1=             X=X+DELX
114. =             DELY=X*Y*DELX
115. =             Y=Y+DELY
118. =             IF(X-1.)3,3,5
120. =           5 PAUSE 1
122. =             GO TO 1
123. =             END
141. +READY
```

FIGURE 9

PROGRAM LISTING

INTERNAL DESIGN

Introduction

Since detailed information on QUIKTRAN's internal design is available[8], only a brief outline is given here. The system is structured into two major sections **(Figure 10)**. The process subsystem translates source statements to an equivalent intermediate representation that is then interpretively executed. The I/O subsystem controls the communications network and the swapping of programs between primary (e.g. core) and secondary (e.g. drum and disk) storage.

Records (Figure 11)

The processor translates all source statements into two types of internal records. The fixed length element records, corresponding to source data and procedural elements, con-

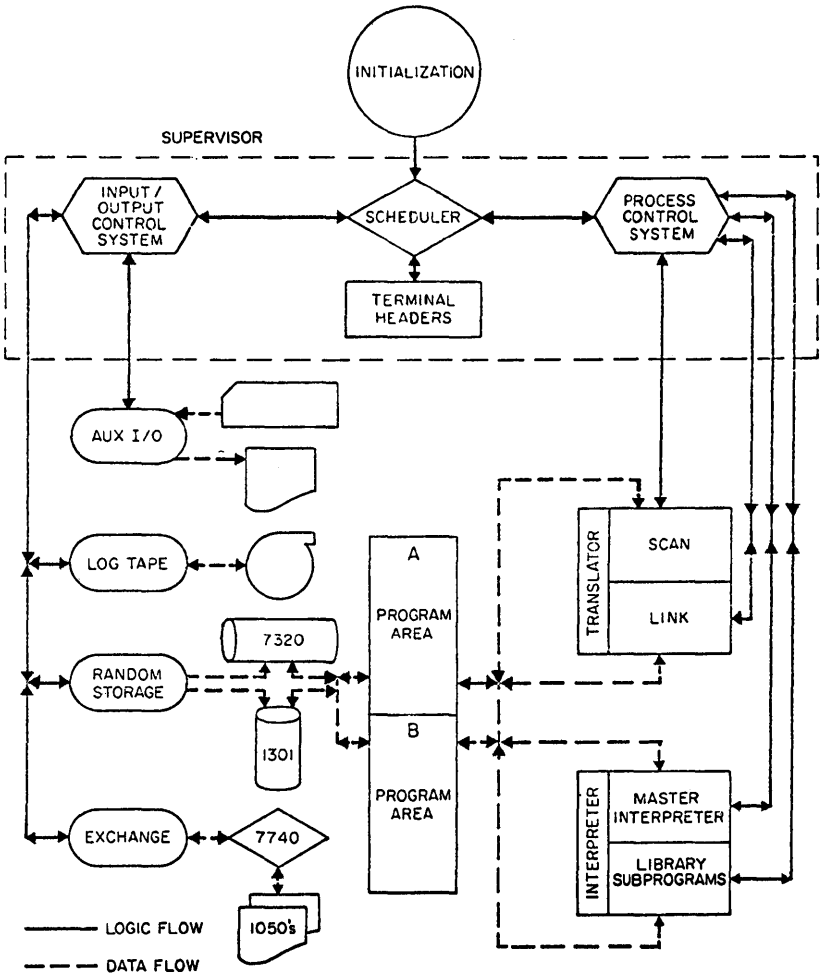


FIGURE 10

INTERNAL ORGANIZATION

tain the usual information collected in the early phases of all compilers (e.g., symbolic name, value attributes, and addresses). In addition, these records contain the value itself and list control information. The variable length statement records, in one-to-one correspondence with source statements, contain the source data in a form that is more compact (to save space) and more suitable for execution (to save time).

Lists

All records are organized by lists. Element records are chained on to one of 26 lists each containing all elements with the same initial letter. This not only reduces symbol search time but also facilitates the generation of alphabetized memory dumps and cross reference listings. All statement records are chained onto two lists: one organizing the statements in entry order; the other classifying state-

Element Id	Type/ Mode	Size	Next Address
Indicators			Dim/Com/Equ Address
Symbolic Name			
Numeric Value			

Element Record

Statement Id		Size	Next Entry Address	
Indicators	Code	Label	Next Class Address	
R(C)	Null		Null	Null
*	R(A)		R(B)	PAROP
R(D)	FUNOP		R(SQRT)	/ T
R(C)	+ TT		← T	R(C)

Statement Record

C = A * B + C/SQRT (D)

FIGURE 11

RECORD FORMATS

ments by type. The former is used to determine proper ordering when reconstructing the source program and when executing the intermediate text; the latter facilitates interstatement error checking.

Addressing

Essentially three distinct types of addressing are employed; associative list searching for symbol matching, look at addressing for mapping internal identifiers to relative locations, and implicit addressing (e.g., pushdowns) for storage of intermediate arithmetic values and for control of DO nesting.

Blocks (Figure 12)

Each user program block consists of two parts; the header contains the control, relocation, and addressing data; the body contains the intermixed element and statement records. Each time a program block is swapped into primary storage, parts of the header are processed to reflect storage relocation and to accumulate usage statistics.

Checking

Checking is performed at four levels. First, composition checking for correct syntax within a statement (e.g. parenthesis imbalance). Second, consistency checking for proper association between statements (e.g., GO TO A FORMAT). Third, completeness checking for proper DO nesting, label referencing, and control flow. Fourth, limit checking for arithmetic spills, proper subscript values, and valid control branching.

Control

A 7740, a separate communications computer, controls the network of remote terminals performing such functions as terminal polling, code conversion, error checking, and buffering of messages awaiting computer attention.

Scheduling (Figure 13)

A section of the 7040 program continually samples a small in-core terminal status table to determine which user program will next receive service.

Logging (Figure 14)

Another 7040 control routine continually records usage statistics on magnetic tape for later off-line analysis and summarization.

EXPERIENCE

The system has been in experimental operation since mid-1963. Early human factor analyses led to some revision in system function and operating procedures[9]. Preliminary evaluation led to the decision in August 1964 to announce the system as a standard IBM program product to be available for customer use

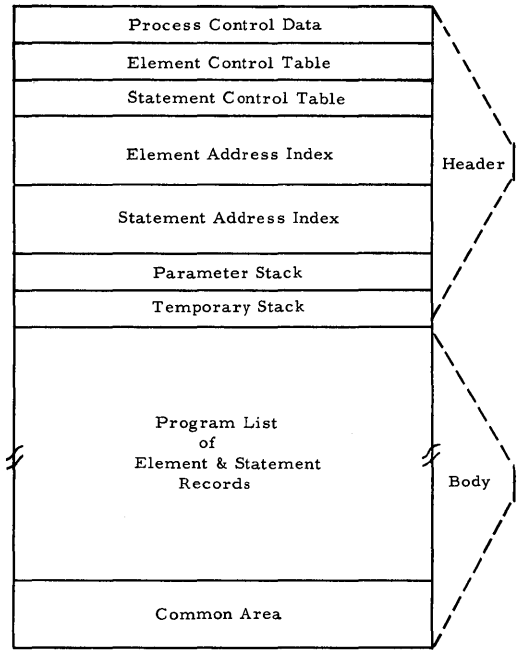


FIGURE 12

PROGRAM BLOCK

Id	Circuit	Type
Id	User Priority	Quantum
Id	Terminal Formats	Components
Id	Job	Time In
Time	Job	Space
System	Indicators	Program
	Program Name	
Size	Program	Location
	Program Calling Program Name	

Circuit Index

Program is evicted when:
1) allotted time interval has expired and successor program has arrived
2) input data is requested
3) output buffer is filled
4) external subprogram is called

FIGURE 13

SCHEDULING

DATA	USE
History	System Recovery
Performance	
Device	Simulation & Billing
Function	Hardware/Software tradeoffs
Language	Specification Changes
Procedure	+
User	Revised Training Methods
Response Rate	
Error Frequency	User Evaluation
Productivity	

FIGURE 14

LOGGING

in April, 1965. Further evaluation led to the December, 1964 announcement of Datacenter access to be available by mid-1965 from terminals installed on client premises.

Initial operating experiences can be partly summarized by the following observations.

The remote user must have the ability not only to state his program but also to control its execution. Thus, many functions conventionally performed by the console operator or by monitor control cards must be identified, structured, and defined by simple, yet comprehensive, language commands.

The remote user is, more often than not, an engineer or scientist, not a professional programmer. Unnecessary sophistication (no matter how elegant) in the system language and operating procedures must be avoided. Remember too, the remote user does not usually have ready access to expert helpers and consultants.

The remote user must be given the impression that he is the only user. All possibility of interference by others must be prevented. The response rate should be reasonably uniform; uneven response fluctuations are particularly disturbing. In addition, some periodic terminal indication of action (e.g., console "blink") helps reassure the user that his program is running. Finally, there must be ways for the user and central operating personnel to communicate both by transmitted messages and also by verbal exchanges.

The terminal itself is a potent training tool. The ability to proceed at one's own pace, coupled with the immediate detection of most programming errors, enables most people to start using a terminal with very little formal training.

Conversational, source language techniques benefit amateur users relatively more than professional programmers who already understand machine language and know how to debug effectively. Experienced programmers are strongly conditioned to traditional batch computing techniques and have considerable trouble in adapting to the conversational approach.

The types of problems run from remote stations differ from those entered at a conventional computer installation. There are many more relatively simple problems previously processed on desk calculators, slide rules, or small computers. Problems with large input/output volumes are obviously not suitable for remote operation. This is also true of many production type problems where object efficiency is important and there is little need for the injection of human insight, judgment, or experience. Conversely, the debugging of complex programs is greatly facilitated not only in terms of elapsed time and expense, but also in terms of the user skill level.

Time sharing systems are very complex, difficult to develop, and challenging to debug. The system must be designed with these factors in mind. In particular, it is essential to provide means of measuring such factors as user and system response rates, use of language features, causes of errors, and equipment utilization factors. This data can be used to simulate alternative system configurations and scheduling algorithms and thereby lead to improved system performance. Analysis of this data is also essential to designing improvements in the language specifications, operating procedures, and training methods. Finally, the data provides a means of equitably allocating overall system expenses among the individual users.

Man-machine systems are no magical panacea. Properly applied they can be very effective; improperly used, they prove to be a very expensive novelty.

EXTENSIONS

Although the QUIKTRAN system uses typewriter oriented consoles, other terminal equipment could be employed: dictation to a remote terminal operator, dialed input with audio

response[10], or graphic input with visual displays[11].

System performance could be greatly enhanced by computer organizations that provide improved interrupt capability, object time address protection and relocation, larger primary storage, and secondary storage equipment with faster access times and data rates.

System performance could also be improved by a program design that permits the intermixed generation of object code along with the interpretive intermediate representation. Execution efficiency of the intermediate code could be improved by incorporating some of the frequently used interpretive functions into micro-programmed logic contained in a read only storage[12].

It is also profitable to explore the translation of several different source languages into the same intermediate form. This would not only reduce implementation time and expense but would also serve as a useful vehicle for translation between related source languages[13].

From a marketing viewpoint, small systems [14, 15] like QUIKTRAN will undoubtedly be absorbed as subsets of larger time sharing systems[16, 17, 18]. However, it is also probable that the stand alone, shared system will continue to provide functional capabilities in the range between the desk calculator and the small computer. Initial use will include the areas of scientific computing, text editing, computer assisted instruction, and certain commercial applications.

CONCLUSIONS

The QUIKTRAN system demonstrates that a standard medium scale computer can be time shared to provide an economical, but not fully general, form of computer service. It is also indicative that effective remote computing requires not only different real time operating systems but also new approaches to translation (e.g., incremental) and debugging (e.g., source language) techniques.

It appears that the system offers little, if any, economic improvement over conventional small computers if measured in established throughput[19, 20] units. However, there are significant advantages in terms of improved convenience, faster turnaround, higher manpower productivity, lower personnel skill levels, and greatly reduced total solution time and expense. Most importantly, such systems not only permit old problems to be solved in new ways, but also enable new users to solve new problems in ways not hitherto possible. It is this factor that will lead to wide utilization of similar man-machine systems in a wide spectrum of applications.

ACKNOWLEDGMENTS

Many individuals contributed to the QUIKTRAN system. The following people made major contributions to its design and development: T. M. Dunn, J. M. Keller, E. C. Strum, and G. H. Yang.

REFERENCES

[1] E. G. Andrews, "Telephone Switching and the Early Bell Lab Computers", *Bell System Technical Journal*, March 1963.

[2] IBM 701 Speedcoding System, IBM Form Number 24-6059, 1953.

[3] IBM 705 Print System, IBM Form Number 32-7855, 1957.

[4] W. R. Brittenhan, et al., "SALE—Simple Algebraic Language for Engineers", *ACM Communications*, October 1959.

[5] Bendix Intercom 1000 System, Bendix Manual CB-029, 1958.

[6] A. Newell (Ed.), *"Information Processing Language—V Manual"*, Prentice Hall, 1961.

[7] IBM 7040/7044 Remote Computing System, IBM Form Number C28-6800, 1964.

[8] J. M. Keller, E. C. Strum, and G. H. Yang, "Remote Computing—An Experimental System. Part 2: Internal Design", *Proceedings of the Spring Joint Computer Conference*, 1964.

[9] T. M. Dunn, and J. H. Morrissey, "Remote Computing—An Experimental System. Part 1: External Specifications", *Proceedings of the Spring Joint Computer Conference*, 1964.

[10] T. Marill, D. Edwards, and W. Feurzeig, "DATA-DIAL. Two-Way Communications with Computers from Ordinary Dial Telephones", *ACM Communications*, October 1963.

[11] G. J. Culler, and R. W. Huff, "Solution of Non-Linear Integral Equations Using On-Line Computer Control", *Proceedings of the Western Joint Computer Conference*, 1962.

[12] J. Anderson, "A Computer for Direct Execution of Algorithmic Languages", *Proceedings of the Eastern Joint Computer Conference*, 1961.

[13] J. J. Allen, D. P. Moore, and H. P. Rogoway, "SHARE Internal Fortran Translator (SIFT)", *Datamation*, March 1963.

[14] S. Boilen, E. Fredkin, J. C. R. Licklider, and J. McCarthy, "A Time Sharing Debugging System for a Small Computer", *Proceedings of the Spring Joint Computer Conference*, 1963.

[15] J. C. Shaw, "JOSS, A Designer's View of an Experimental On-Line Computing System", *Proceedings of the Fall Joint Computer Conference*, 1964.

[16] F. J. Corbato, M. Merwin-Daggett, and R. C. Daley, "An Experimental Time Sharing System", *Proceedings of the Spring Joint Computer Conference*, 1962.

[17] E. G. Coffman, J. I. Schwartz, and C. Weissman, "A General-Purpose Time Sharing System", *Proceedings of the Spring Joint Computer Conference*, 1963.

[18] H. A. Kinslow, "The Time Sharing Monitor System", *Proceedings of the Fall Joint Computer Conference*, 1964.

[19] R. L. Patrick, "So You Want To Go On-Line", *Datamation*, October 1963.

[20] "A Panel Discussion on Time Sharing", *Datamation*, November 1964.

VI

EXAMPLES AND SUMMARY

Glen D. Johnson*

THE PAT LANGUAGE

THE SYSTEM to be described is an on-line interpreter for a structured, algebraic language. This interpreter is operating on the UCLA Computing Facility 7094 with the SWAC computer used to maintain a typewriter console. There is also a similar interpreter for the IBM 1620. The 1620 version was produced by Dr. H. Hellerman of the IBM Advanced Systems Development Division in Yorktown Heights, New York, where it has had several hundred hours of productive use.

The language is called the Personalized Array Translator—just PAT for short. The PAT language is a subset of the IVERSON language which was designed by Dr. Kenneth Iverson of IBM as a mathematical description tool. The character set used by Iverson has been reduced to a manageable size in the PAT implementation.

A program in the PAT language operates on data which is highly structured. The basic data structure is considered to be a vector. Each symbolic name is specified by the system's operator to be either a scalar, a vector, or a matrix. Data names specified to be matrices or vectors also must have their maximum sizes specified. There are statements in the language to access and modify the sizes of variables.

Each variable is stored by the system as a vector with matrices represented as a series of column vectors stored in the computer. Scalar data items are represented as vectors of unit length.

Currently, the 7094 system allows Boolean and floating point data items. Floating point constants are represented as numbers with or without a decimal point. False is represented by 0, true by 1. A constant may appear anywhere that a variable is allowed.

PAT allows portions of data items to be operated on using a subscripting rule. All subscripting is zero-origin with X_0 as the first element of X. A single element may be selected from a vector by using one subscript or from a matrix by using two subscripts. A vector, either row-wise or columnwise, may be selected from a matrix by giving one subscript with an indication that the other subscript is empty.

For example, let S be scalar, V be a vector, and M be a matrix. Then:

VS	is a scalar
M S1 S2	is a scalar
M S	is a row vector
M . S	is a column vector
A .	indicates an empty position

Most statements are of the form:

Variable=Expression

They have the conventional meaning: the value of the expression is computed and stored in the variable on the left.

This meaning is extended by allowing variables to have more than one element. An expression is evaluated using the first element of every variable contained in it to form the first element of the resulting variable. The expression is then evaluated using the next element of each argument to form the next element of the result, and so on until the resulting variable is filled.

For example, if

X=1, 2, 3, 5

and

Y=7, 11, 13, 17

*UCLA Computing Facility

and the current size of Z is 4, after execution of

```
Z=X+Y
```

the value of Z is

8, 13, 16, 22

If the end of any variable in an expression is reached before the left part of the statement is filled, indexing on this variable is restarted at its first element.

This may be formalized by considering each operand, having a length L to be periodic of period L. The interpreter stores the first cycle of each variable and generates later cycles as copies of the first cycle.

There are three types of data combination operations allowed by the PAT system. They are binary operations, unary operations, and reduction operations. Binary operations are represented in infix form as an operator between two variables and include *addition, subtraction, multiplication, division,* and *maximum, minimum, and, or,* and *relational* operators.

BINARY STATEMENTS

```
X=Y+Z
X=Y-Z
X=Y*Z
X=Y@DIV Z
X=Y@MIN Z
X=Y@MAX Z
X=Y@EXP Z
```

Note: Operation names are preceded by "@", and only the first character of name is used.

Unary operations include trigonometric and logarithmetic operations.

UNARY STATEMENTS

```
X=@SINE Y
X=@COS Y
X=@ABS Y
X=@LN Y
```

Reduction operations are binary operations across data structures. Summation is a specific case of reduction. In general, any binary operation is applied to a vector or a matrix as:

+ / Vector	Summation
Θ / V	Generalized
$V_0 \Theta V_1 \Theta V_2 \ldots V_n$	
Θ / Matrix	Row Reduction
Θ / / Matrix	Column Reduction

For example, to compute:

$$\bar{x} = \frac{\sum_{0}^{n} x_i}{n} \qquad s = \frac{\sum_{0}^{n} (x - \bar{x})^2}{n}$$

we write:

```
XB=+ / X          sum X
XB=XB @DIV N      over N
T  =X-XB          differences
T  =T * T         squared
S  =+ / T         summed
S  =S@DIV N       over N
```

To read and print data for this, we write:

```
@GET   N
@DIM   X,N
@D     T,N
@G     X
{      }
@TYPE  XB
@T     S
```

A statement line may be labelled. If a line is given a name, the name starts in the first position of the line. Otherwise, the first position is blank.

Normal sequencing of a program is from top to bottom, left to right. Sequencing may be changed and explicit looping effected with a compare statement.

```
      I=0
L     ZI . = X . I
      I = I + 1
      I @CMP N, L, L, O
```

The console has a series of commands used to communicate with the program. These have a * in the first typed column of a line. They are:

*R	Reset system
**	Define symbols, followed by

(name{Scalar / Vector / Matrix}{Floating / Boolean}[dim1][dim2])

*E [name]	Enter program statements (program statements)
*X [name]	Execute program
*D name	Display

David V. Savidge*

An Example of Multi-Processor Organization

INTRODUCTION

As the purpose of this meeting is the free exchange of ideas, it is necessary to establish the means of communication by defining the terms in this title. We hope to continue on-line throughout this discourse.

The justification for the inclusion of this subject in this symposium is the fact that at some point in time someone will put together a system to handle a multiplicity of problems on-line. Before programs can be run on such a system they must be organized in such a manner as to facilitate their execution. By exploring ways to organize programs now we will be better able to utilize the hardware when it becomes available.

DEFINITIONS

The kind of example we will show is that of a method. References to hardware will only be used to demonstrate feasibility and the fact that the ideas discussed are not novel. The intent of this paper is to show a method which presents some interesting possibilities for further exploration.

Before defining multi-processor it is necessary to accept a definition of a processor. We consider a processor to be an assembly of hardware which is capable of performing one or more arithmetic or logical functions in a specified manner. By this definition the word processor could describe a device which only performs addition. It could also be applied to the computing unit of the LARC. This leads to a definition of a multi-processor as being any mixture of processors which share one or more components such as memories, input devices or output devices. This permits us to include the MARK II[1] and the Bell Labs MOD V[2] in the category of multi-processors as well as the LARC itself. Each of the above contains two processors by our definition. A more complex array is presented by the ENIAC as described in Patent Number 3,120,606, granted Feb. 4, 1964. The ENIAC contained twenty accumulators. In addition to operating on two problems concurrently, the ENIAC could perform several operations at the same time on one problem. All of these systems are considered, in this paper, to be multi-processors.

Whenever two or more pieces of anything are put together they must be organized. Certain customs or environmental conditions often impose constraints on the organization. Even though the engine can be found in either the front or the back of an automobile, the driver's seat remains in the front. A horse is generally placed in front of the cart it is intended to pull. The element of direction or control is integral to any organization of pieces required to do work. This paper will describe how the control of a multi-processor can be set up to provide the response times needed in closed loop applications as well as the generalized treatment required in time sharing systems. We are not concerned with the capability of the individual processors but rather with the broad question of the organization of work to be performed by a multiplicity of processors. We do not consider that

*Manager, Product Line Planning, UNIVAC.

the assignment of one procedure to one processor while other processors remain idle represents efficient utilization of the hardware.

The idea of a system comprising a multiplicity of processors seems to be a natural extension of the time sharing concept. Time sharing was the outgrowth of the imbalance between CPU and I/O device speeds. If the CPU was loafing, it was given more work to do. We now have the line capability to overload one CPU within an installation. The logical extension is to provide more than one CPU. The question immediately arises—what does one do if there is only one problem to be run at this time? Is it satisfactory to use only one of the CPU's which are in the system?

Various schemes have been proposed for the design of multi-processors of varying capabilities—the Holland system[3] and the Solomon computer[4] are examples of this. Concepts of control are being explored by many research groups. The application of NDRO memories is increasing. A similarity is noted between the use made of NDRO memories and the utilization of the function tables of the ENIAC[5]. The organization of the ENIAC permitted the programmer to organize the solution of a problem so that more than one arithmetic operation was being performed at one time. This was difficult, but was one way to shorten the execution time for a problem. We are faced, today, with the same logical problem as faced ENIAC programmers. The difference lies in the fact that we now have a variety of gear and a multiplicity of problems brought together in a multi-dimensional array. It is our thesis that it is possible to automate the organization of a single procedure to maximize the utilization of multiple processors.

Unless the organization of the procedure is performed according to a very rigid set of rules it will provide another source of subtle errors. While it is assumed that all parts of a stated procedure are interrelated within the total network, it does not follow that all steps must be performed in series. One of the ways in which processors gain speed is to overlap input, processing and output. We now want to extend this philosophy to the internal parts and determine the extent to which overlaps can occur within the process. Some equipment provides "look ahead" which permits the overlapping of the time of instructions which occur in series. This is accomplished within a single processor. When dealing with a multiplicity of processors similar functions could be performed in parallel.

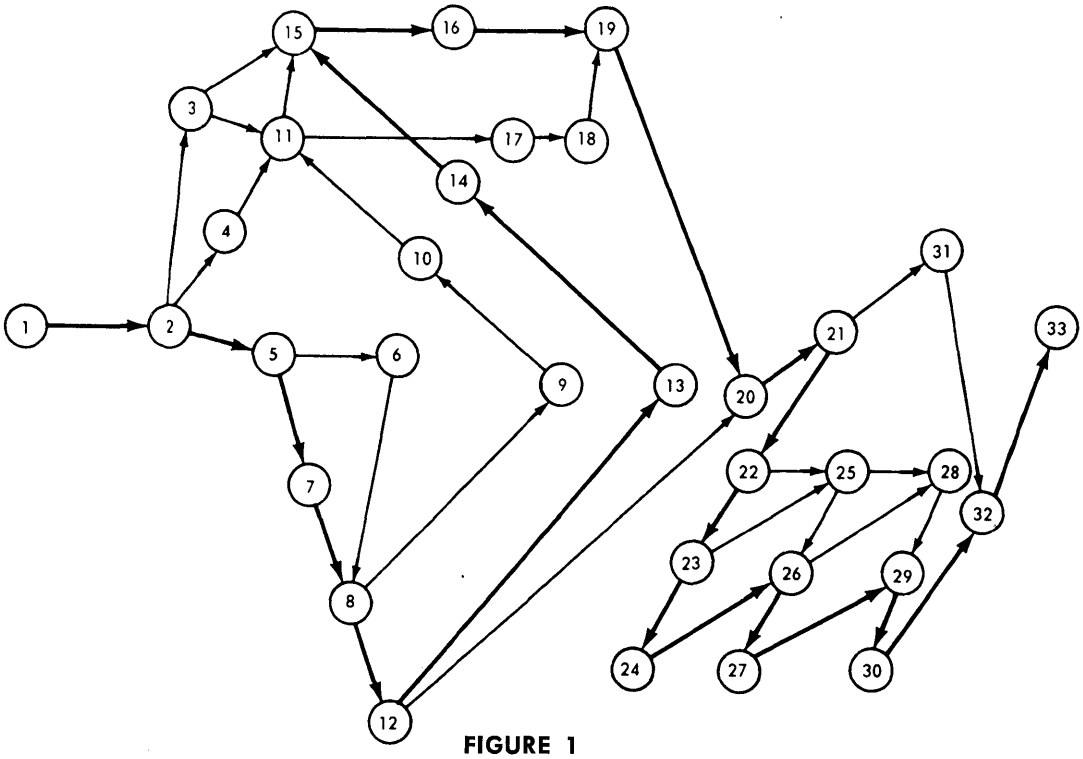


FIGURE 1

PERT NETWORK

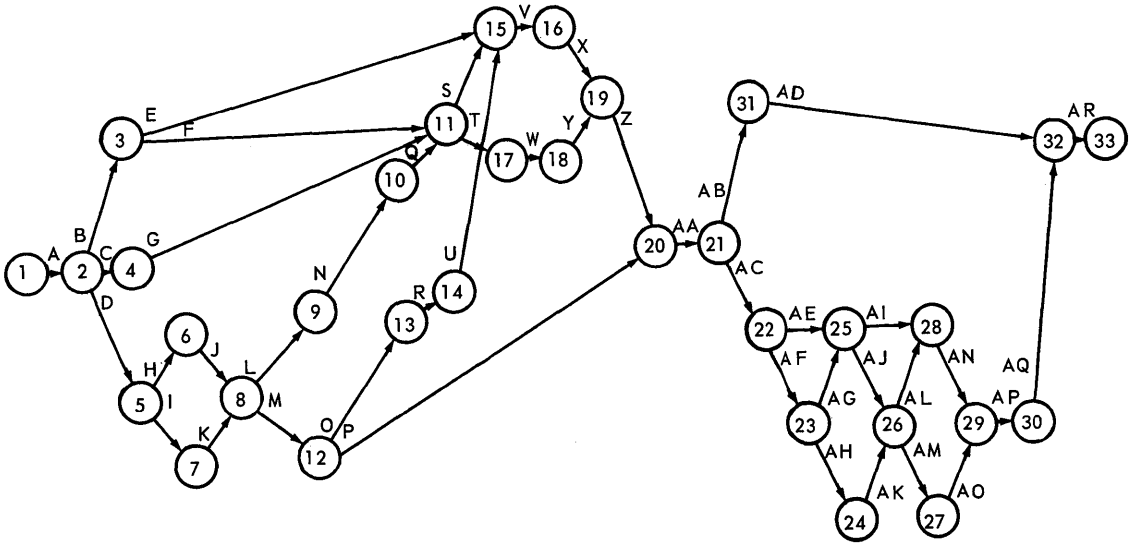


FIGURE 2

PERT NETWORK ARRANGED BY TIME PERIOD

This can be achieved at the software level by what we choose to call "plan ahead". The organization can be accomplished by the compilers and the executive routines.

We must make several basic assumptions and accept certain definitions in order to develop a set of rules:

1. A procedure is defined as the collection of operations required to produce specified output data from specified input data.
2. A procedure generally consists of a set of subprocedures which are linked together in the form of a network.
3. An individual subprocedure can be defined as a completely self contained process with a prescribed set of inputs and outputs.
4. The communication between one subprocedure and any others occurs only at the beginning and at the end of the subprocedure.
5. It is possible to depict the flow of data by means of a diagram which shows the interrelationship of all subprocedures within the procedure.
6. The flow diagram can indicate those subprocedures which could be executed in parallel.

PROCEDURES

Figure 1 and **Figure 2** demonstrate an application of these definitions. **Figure 1** is a PERT chart of a procedure[6]. Each of the 33 numbers represents a subprocedure. **Figure 2** shows the same flow diagram arranged to indicate the parallelism possible. In **Figure 2,** the lines connecting the subprocedures have been identified with letters which we will use to denote the data (operands) flowing between the subprocedures. Twelve of the subprocedures fall into time periods by themselves (1, 2, 8, 19, 20, 21, 23, 26, 29, 30, 32, 33). Eighteen can be paired:

6, 7 9, 12 10, 13 11, 14
15, 17 16, 18 22, 31 24, 25 27, 28

One group could contain three (3, 4, 5). If we shifted either 3 or 4 to occur in the same time period as 8, a two processor system could accomplish the entire procedure in twenty-two time periods. The reduction in time over a serial operation would be the sum of the times for the shorter of each of the eleven pairs of subprocedures. One of the processors would be available for the execution of another procedure during the eleven time periods when the illustrated procedure does not permit parallel execution of its subprocedures.

This example is used to indicate the method by which a reduction in elapsed time could be achieved. The evaluation of any saving requires additional specifications such as the time for each subprocedure. This is related to the problem and the hardware. The method is independent of the specifications of either the problem or the hardware. The first question to

be resolved is whether the problem lends itself to parallel operation. The second is to measure the savings which could be achieved.

An improvement in running time can be achieved by proper pairing (in a two processor system). Depending on how nearly the time requirements match, the pairing of 5 with 4 and 3 with 8 might be better than pairing 5 with 3 and 4 with 8. Similarly, 31 should be paired with an operation which requires more time than it does. This could be 22, 23, 26, 29, or 30. The rule is that a subprocedure which could be performed in one of several periods should be paired where possible with a subprocedure which required more time than it does . If not possible, the longest fixed position subprocedure should be used.

A further consideration in selecting time periods in a complete procedure is the storage of intermediate results. To shorten the period for storing intermediate results it might be better to perform subprocedure 3 in parallel with subprocedure 8, even though 3 is shorter than 5 and longer than 8. Such considerations are of value only where there are alternatives. The fact that alternatives do exist is evident from a visual examination of **Figure 2.**

If the procedure were large, of several thousand subprocedures, a visual representation of the entire net would be very difficult to prepare. It would also be difficult to examine and analyze. It is possible to represent the intelligence represented by **Figure 2** in a form which can be used for processing by a computer. **Table 1** contains this information. List I shows one entry for each operand result relationship arranged in order by subprocedure identification. List II is the same data arranged in order

TABLE 1

SUBPROCEDURE ENTRIES FOR ANALYSIS

List I			List II			List III			List I			List II			List III		
SP	O	R	O	R	SP	R	O	SP	SP	O	R	O	R	SP	R	O	SP
2	A	B	A	B	2	AA	P	20	17	T	W	D	H	5	F	B	3
2	A	C	A	C	2	AA	Z	20	18	W	Y	D	I	5	G	C	4
2	A	D	A	D	2	AB	AA	21	19	X	Z	E	V	15	H	D	5
3	B	E	AA	AB	21	AC	AA	21	19	Y	Z	F	S	11	I	D	5
3	B	F	AA	AC	21	AD	AB	31	20	P	AA	F	T	11	J	H	6
4	C	G	AB	AD	31	AE	AC	22	20	Z	AA	G	S	11	K	I	7
5	D	H	AC	AE	22	AF	AC	22	21	AA	AB	G	T	11	L	J	8
5	D	I	AC	AF	22	AG	AF	23	21	AA	AC	H	J	6	L	K	8
6	H	J	AD	AR	32	AH	AF	23	22	AC	AE	I	K	7	M	J	8
7	I	K	AE	AI	25	AI	AE	25	22	AC	AF	J	L	8	M	K	8
8	J	L	AE	AJ	25	AI	AG	25	23	AF	AG	J	M	8	N	L	9
8	J	M	AF	AG	23	AJ	AE	25	23	AF	AH	K	L	8	O	M	12
8	K	L	AF	AH	23	AJ	AG	25	24	AH	AK	K	M	8	P	M	12
8	K	M	AG	AI	25	AK	AH	24	25	AE	AI	L	N	9	Q	N	10
9	L	N	AG	AJ	25	AL	AJ	26	25	AE	AJ	M	O	12	R	O	13
10	N	Q	AH	AK	24	AL	AK	26	25	AG	AI	M	P	12	S	F	11
11	F	S	AI	AN	28	AM	AJ	26	25	AG	AJ	N	Q	10	S	G	11
11	F	T	AJ	AL	26	AM	AK	26	26	AJ	AL	O	R	13	S	Q	11
11	G	S	AJ	AM	26	AN	AI	28	26	AJ	AM	P	AA	20	T	F	11
11	G	T	AK	AL	26	AN	AL	28	26	AK	AL	Q	S	11	T	G	11
11	Q	S	AK	AM	26	AO	AM	27	26	AK	AM	Q	T	11	T	Q	11
11	Q	T	AL	AN	28	AP	AN	29	27	AM	AO	R	U	14	U	R	14
12	M	O	AM	AO	27	AP	AO	29	28	AI	AN	S	V	15	V	E	15
12	M	P	AN	AP	29	AQ	AP	30	28	AL	AN	T	W	17	V	S	15
13	O	R	AO	AP	29	AR	AD	32	29	AN	AP	U	V	15	V	U	15
14	R	U	AP	AQ	30	AR	AQ	32	29	AO	AP	V	X	16	W	T	17
15	E	V	AQ	AR	32	B	A	2	30	AP	AQ	W	Y	18	X	V	16
15	S	V	B	E	3	C	A	2	31	AB	AD	X	Z	19	Y	W	18
15	U	V	B	F	3	D	A	2	32	AD	AR	Y	Z	19	Z	X	19
16	V	X	C	G	4	E	B	3	32	AQ	AR	Z	AA	20	Z	Y	19

Legend
SP–Subprocedure
O–Operand
R–Result

by operand identifier. List III is the same data arranged in order by result identifier. As subprocedure 1 and subprocedure 33 do not each have an input and an output, they are not included in the lists. By truncating List I at different points and rearranging the remainder into Lists II and III, a variety of combinations can be produced.

Subprocedure 2 generates three table entries. Subprocedure 11 generates six entries, while subprocedure 16 generates only one entry. The rule is that each subprocedure generates a number of table entries equal to the product of the number of inputs times the number of outputs. Each entry must appear in each of the three tables. To facilitate scheduling, each entry should carry additional data concerning the facilities used by the subprocedure, the input and output volumes involved and the time required. If this is done the complete network can be timed out, scheduled, and controlled for any combination of processors.

Before describing the techniques to be applied to the three lists and the results which can be secured, it is necessary to delimit the terms further.

1. An operand is all of the data which flows from one subprocedure, or an input, to another subprocedure, or an output, in one movement as one record or piece of intelligence. Thus it can be an element of data, as a quantity to be added, or a set of data as a stock record. It must be received from some point outside the subprocedure which operates on it.
2. Parameters which are used by a subprocedure are not considered to be operands within this definition. This does not preclude their use in arithmetic or logical operations within a subprocedure.
3. The definitions of individual subprocedures, operands and parameters are always unique within a given environment consisting of problems and hardware.

The treatment of necessary prior conditions will be discussed later. We seldom go through all paths in all subprocedures for one record or piece of intelligence. Conditions which must be met before executing an individual path represent intelligence which is derived from the data processed. There are options in the way such conditions are treated, depending on the problem and the hardware. For this reason a discussion of their treatment is deferred.

ORGANIZING PROCEDURES

The technique outlined below (Steps 1 through 6) produces a list which represents the same time series as shown in **Figure 2.** If the operation is started with the final results (Steps 1 through 6A), a list is produced which shows the last possible time period by which subprocedures must be executed.

Step 1—Compare the Operand Fields in List II with the Result Fields in List III. Note four conditions:

a. O Field in List II does not match an R Field in List III—
Record items on a list of Unmatched Operands—

O	R	SP
A	B	2
A	C	2
A	D	2

This condition must be noted for each relative time period when analyzing the subprocedures for first possible time period. It will be ignored when analyzing for the last possible time period.

b. O Field in List II does match an R Field in List III—
Record items on a list of Matched Operands. This is a reduced List II.

c. R Field in List III does match an O Field in List II—
Record on a list of Matched Results. This is a reduced List III.

d. R Field in List III does not match an O Field in List II—
Record on a List of Unmatched Results.

R	O	SP
AR	AD	32
AR	AQ	32

This condition will be ignored when analyzing for the first possible time period. It must be noted for each relative time period when analyzing for the last possible time period.

Series for first possible time period—

Step 2—Rearrange the data from Step 1a to read—

R	O	SP
B	A	2
C	A	2
D	A	2

Sort on the R Field.

Step 3–Delete the items from the List of Matched Results (Step 1c) which are identical to those from Step 2. If this deletion removes all references to the result identifiers we state that these results can be secured in the first relative time period.

Step 4–Rearrange the data from Step 2 to read–

SP	O	R
2	A	B
2	A	C
2	A	D

Sort on the SP Field.

Step 5–Delete the items from List I which are identical to those from Step 4. If this deletion removes all references to the subprocedure identifiers we state that these subprocedures can be accomplished in the first relative time period.

Step 6 and continuing–Repeat Steps 1 through 5 for successive time periods until Lists I, II and III are exhausted.

Series for last possible time period.

Step 2A–Rearrange the data from Step 1d to read–

O	R	SP
AD	AR	32
AQ	AR	32

Sort on the O Field.

Step 3A–Delete the items from the List of Matched Operands (Step 1b) which are identical to those from Step 2A. If this deletion removes all references to the Operand identifiers we state that these operands must be used by the last relative time period.

Step 4A–Rearrange the data from Step 2A to read–

SP	O	R
32	AD	AR
32	AQ	AR

Sort on the SP Field.

Step 5A–Delete the items from List I which are identical to those from Step 4A. If this deletion removes all references to the subprocedure identifiers we state that these subprocedures must be accomplished by the last relative time period.

Step 6A and continuing–Repeat Steps 1, 2A through 5A for preceding time periods until Lists I, II and III are exhausted.

Unusual conditions can be detected by this technique. The operands which are initial inputs, and the results which are final outputs, are identified as the lists secured in Step 1a and 1d the first time. Any errors in their identification are corrected before performing the other steps. In addition, if items remain in Lists I, II and III and no items fall out in step 1a or 1d as the case may be, a closed loop exists which must be corrected.

Table 2 shows the results of this analysis by relative time period. The fact that subprocedures 3, 4 and 31 can each be scheduled in one of several relative time periods is evident. The selection of the best fit can be accomplished on a computer by applying the rules stated previously. One of the advantages which can be gained by using a multiprocessor is the reduction in the time and effort required to store and retrieve intermediate results.

The above technique organizes the subprocedures which produce results. For each relative time period, that subprocedure which requires the greatest amount of time can be identified. The complete chains of subprocedures which can be assigned in series to one processor can be identified. The exchange of data between processors can be scheduled to minimize memory requirements. These are positive benefits which can be achieved with this technique. It applies a modification of PERT and CPM to the organization of work for a computer.

CONDITIONS

The treatment of necessary prior conditions can be accomplished with the same technique. We need only to regard a comparison, which determines the path to be followed between subprocedures, as generating data. For our purpose we treat intelligence about data in the same way we treat the data itself. It is only necessary to identify this intelligence in the same way we identify data and then proceed through the same steps. We can identify intelligence about data with the form:

Operand 1, Operand 2, Value.

Operand 1 and Operand 2 can be data fields or can be literals. Any data fields must be shown as an input to the subprocedure performing the comparison. The value field is considered necessary because we can never have only one such result coming from a subprocedure. The value field permits a verification that all conditions have been considered.

TABLE 2

ALLOCATION OF SUBPROCEDURES

Relative Time Period	First possible execution of Sub-Procedure	Last possible execution of Sub-Procedure
1	2	2
2	3,4,5	5
3	6,7	6,7
4	8	8
5	9,12	9,12
6	10,13	3,4,10,13
7	11,14	11,14
8	15,17	15,17
9	16,18	16,18
10	19	19
11	20	20
12	21	21
13	22,31	22
14	23	23
15	24,25	24,25
16	26	26
17	27,28	27,28
18	29	29
19	30	30,31
20	32	32

The sum of all value fields must equal seven for each comparison.

TABLE 3

CONDITION VALUES ON COMPARISONS

Condition	Value
$=$	1
$<$	2
$>$	4
$\neq$	6
$\leq$	3
$\geq$	5

From the foregoing it is obvious that the simplest technique to use for comparisons which affect branching between subprocedures, is to treat the comparisons as subprocedures by themselves. Each decision affecting branching would be a subprocedure. This leads us to an alternative method. The main flows of data in a multi-processor could be executed without regard to data dependent branching. All data dependent decisions could be made by a separate processor and the proper final results selected. By this method some operations would be performed which would prove useless. Depending on the environment this alternative might save considerable elaped time.

COMPILING AND EXECUTING

We assume that the organization of the procedure will take place prior to the compilation of an object program. The compiler should be able to provide, with the loadable program, all of the intelligence concerning the network, the time periods and the execution times for each subprocedure. In turn the executive or monitor routine must be able to treat each scheduled subprocedure in the same manner as it treats any interrupt from an outside source. A subprocedure would take on the priority of its governing procedure. In effect we are subdividing every piece of work to be done into the smallest practical unit. A subprocedure could be the inversion of a matrix or the comparison of two data fields. The compiler determines how the work can be subdivided and over-lapped. The executive routine determines which component shall execute it and when.

HARDWARE

To provide complete flexibility in the hardware associated with the central memory, a binary addressing scheme seems to be the logical choice. It also seems desirable to provide one static register per processor with a bit size equal to the memory width. Each program would be assigned a base register which would contain a binary number, the starting bit address. Each processor can contain a one, two, or three bit multiplier (hardware) which would operate on the address portion or portions of an instruction before incrementing with the base register. If a processor contains only two static registers and a plugboard, all intelligence as to sequence must be within another processor, possibly the one which contains the executive routine.

CONCLUSION

We have described a technique to break down the solutions of problems to permit maximum parallel operation within one problem. We have not described in detail the hardware needed to execute programs in this manner. We consider that the hardware will be developed.

REFERENCES

[1]Staff of Engineering Research Associates, Inc., *High Speed Computing Devices,* McGraw Hill Book Co., Inc., New York, 1959, p 185.

[2]Ibid, p 188.

[3]J. H. Holland, "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously", *Proc. Eastern Joint Computer Conference,* (1959).

[4]D. L. Slotnick, W. C. Borck, and McReynolds, "The Solomon Computer", *Proc. Fall Joint Computer Conference,* (1962).

[5]Martin H. Weik, "The ENIAC Story", *Ordnance,* American Ordnance Association, Washington, D. C., January-February 1961, p 571-575, also Ref. 1, p 194-197.

[6]Reston M. Aras, ASCE, and Julius Surkis, "PERT and CPM Techniques in Project Management", *Journal of the Construction Division, Proceedings of the American Society of Civil Engineers,* Vol. 90, No. CO1, March 1964, reproduced in UNIVAC publication UP3952.

Dr. Harry D. Huskey*

On-Line Computing Systems: A Summary

ON-LINE COMPUTING was defined by Walter Bauer as the efficient use of a computer in a system in which the computer interfaces with man or other machines, to which it reacts in receiving and supplying information. Reviewing the six kinds of on-line systems described by Ivan Sutherland, I would propose the following revision of the definition:

> "On-line computing is the use of a computer interfaced with man or machines, to which it reacts in receiving, processing, and transmitting information."

I am most interested in the interface with man, and here the computer is on-line if the man awaits the computer's response, not having turned his attention to other matters.

It is clear that during the era 1949 until, perhaps, 1954, all automatic computers were essentially used in an on-line manner. In other words the customer sat at the console, pushed buttons, and tried to interpret the displays (binary neon indicators) in terms of the expected progress of the problem.

In 1954 with the delivery of large commercial computers (with more attention to actual dollar costs) it was no longer sensible to let a user flounder through a problem by pushing buttons and "thinking" at a console. Thus, we have the era of BATCH PROCESSING.

Now in 1965, using the experience gained in developing military systems, many efforts are underway to improve the communication between man and the computer. Obviously the man will make maximum progress on his problem if he can work on it with sustained attention. This implies the use of a personal console available to the man for relatively long periods of time. In most situations his optimum progress on his problem requires computing over very short intervals of time. Consequently, a well organized central computer capable of servicing many low cost stations is required.

ECONOMICS OF ON-LINE COMPUTING

At the University of California in Berkeley 16,000 problem passes were made in the month of January, 1965, on the DCS (7040-7094) system utilizing 57 percent of the available computer time. The average problem time was 1.56 minutes and the average equipment cost per problem-pass was $3.00. The cost of expendable supplies per problem-pass was about $0.30.

The capital cost of a teletype style station ranges from $400 to $2,000, and communication costs range from $2.00 per month (for direct lines on the Berkeley campus) up to $4,300 per month for a leased private voice-grade line across the United States.

Since the teletype user can see the immediate effect of his editing of his program he will use substantially more central computer time. I estimate three times as much which I think is a conservative figure.

Certainly the use of the on-line stations reduces the use of cards and high speed printer supplies. Let us assume that this goes to zero

*Professor of Electrical Engineering, University of California, Berkeley.

and that the cost of paper and ribbons for teletypes can be ignored. If a teletype station is used 300 hours per month (14 hours per day for a five day week) and costs $150 per month (the price of a QUIKTRAN station – note that Morrissey quotes $1,000 per month per station as total cost), then the remote-station hardware cost per problem is $0.50 on the basis of one hour use. Central computer use per problem is probably up by a factor of three, e.g. instead of three compilations to debug a ten statement problem there is perhaps eight to ten passes.

Therefore, the actual cost of doing a problem using remote stations on the basis described above is perhaps three times the cost by batch processing methods, primarily because of the extra problem-passes which can be easily taken from the console.

Certainly reduction in cost of in-output equipment at the central computer, improved methods such as incremental compiling (statement by statement), incentives to encourage non-wasteful use, can improve the cost ratio. However, even under optimum arrangements the total problem cost for remote station computing appears to be perhaps twice that for batch processing.

Those who argue that remote station computing will be cheaper, do so on the basis that (1) incremental compiling will permit clearing syntax errors in one pass (most current batch processing compilers could be much better designed from this point of view), and (2) that fewer results will be printed since the person with the problem knows he can easily get other results if needed. I am willing to admit that there will be a reduction in the printing of wrong results, but at reasonable traffic levels the person with a problem to be solved does not easily get other results. Consequently, when he thinks his program is correct (and he is an eternal optimist) he will ask (curbed only by hard cash economics) for extensive print-outs at the central computer facilities.

It is unfortunate that the problems of saturation of the on-line console system were not discussed.

Whatever the true picture relative to the costs of batch processing versus console computing, there is no doubt whatsoever that from the point of view of elapsed time the console is far superior. As Weizenbaum says: Man is conserved, not the machine.

On the other hand, the system designer has a tremendous task in discouraging the user from experimenting trivially just to see what wonderful things will be done for him.

It is exactly this last point and the economics discussed above that leads us at Berkeley *not* to plan for consoles for undergraduate teaching purposes. In fact, these students do not even get normal turn around. If their problems are left by a fixed time in the evening the results will be back the next morning. This system permits the assignment of a substantial problem once a week due one week later.

At the same time we are installing remote consoles (1050's) which, during scheduled periods provides QUIKTRAN, and at all other times permits communication with problems in process in the DCS system (problems running under IBSYS, FORTRAN II monitor, and numerous special languages such as COBOL, COMIT, SNOBOL, IPL, NELIAC, etc.). Such facilities will be available to graduate students and staff but *not* in undergraduate teaching.

FUTURE OF ON-LINE COMPUTING

Walter Bauer has estimated that by 1975, 90 percent of computing will be done ON-LINE. I would like to examine this for a moment. Following Ivan Sutherland we can divide computing into the following categories:

1. Process control
2. Inquiry Stations
3. Process control
   Specialized Systems (Engineering Design)
4. Instrumentation of on-line systems
5. Programming Systems
6. Problem Solving Systems

Categories one through four are intrinsically on-line. The tasks associated with five and six can be accomplished by batch processing or on-line techniques. My opinion is that (because of economics and, in line with the discussion of the preceding section) program debugging and problem solving will always be done both on-line and by batch processing. In these two areas the balance is anyone's guess—it depends strongly upon the relative accomplishments in the improvement of batch processing techniques and in the improvement of on-line techniques. My guess is that the division will be

about 50%-50%. It is possible that in 1975 process control, inquiry stations, and on-line instrumentation will represent 80 percent of the total computing done, in which case Walter Bauer's estimate would be right. However, I believe that more than 10 percent of the total computing will still be batch processing.

INSTRUMENTATION

The evidence of the lack of precise information in the two preceeding sections supports the contention of Sutherland and Amdahl that much more instrumentation of on-line systems is needed so that we know what is going on, what the typical user does, and what the variations are from the norms. It is only with this information that systems can be "trimmed" so as to optimize usefulness to the customer array.

TYPES OF COMPUTER USERS

Computer users can be classified into two groups: Those who require hardware response before loss of attention occurs or frustration cancels any advantage, and those who only need results some time later (minutes, hours, or days) as they can, without ultimate loss, turn their attention to other tasks in the intervening period. The on-line user is thus distinguished from the batch processing user.

It is also possible to classify users on another scale as follows:

1. The *query* user, who asks the machine (hardware and software) questions and expects answers in real-time (he awaits the answer without transferring attention to other matters).
2. The *reaction* user, who submits himself to the control of the machine. Usually the decisions involved depend upon an environment too complex and too quickly changing for a man (or team of men) to make the decisions. The answers to all questions are programmed *a priori*. An example is the abortion program of project Mercury.
3. The *heuristic* user. A complex problem is under consideration, and the complete exploration of the solution tree is beyond the capability of the system (probably time-wise). Therefore, on some basis the solution tree is pruned by the man and the machine moves him through the reduced tree.
4. The partnership. The partnership between man and the machine can range from a limited partnership as represented by Morrissey's QUIKTRAN to a more general partnership represented by project MAC at MIT, or the time sharing system at SDC. The work on self-organizing systems is looking forward to even more interesting partnerships.

FAIL SAFE

On-line systems are still in their early development stage, but now that systems are beginning to work, I think that it is obvious that more attention should be paid to the fail safe aspects of the problem. Topics to be considered here are memory protection or assignment, system controlled input and output, and well designed user language features. The merits of simpler systems, like JOSS, should not be discounted. Walter Bauer made a significant point in this respect: the user should be led down a procedural path.

THE CULLER-FRIED APPROACH

The use of a storage tube simplifies the remote station device, placing it in the same cost area as sophisticated teletype-style stations. Although such stations are not as versatile as the dynamic or memory buffered types, they nonetheless accomplish the most significant improvements in communication that display scopes represent. In fact, the comments of Pope and Tomkins support the old adage: a picture is better than a thousand words.

As indicated by Pope, perhaps the most important aspect of such station systems is their aid in the development of the solution algorithm. In comparison, teletypewriter methods might be like trying to appreciate the art in the National Gallery by exploring it at night with a pencil size flashlight.

ENGINEERING DESIGN WITH DISPLAYS

The use of display consoles in engineering design seems to me (an outsider) to be pushing the state of the art a bit. My feeling is that here is a powerful technique with almost unlimited potentialities and, therefore, I strongly support all the work in the field. However, at this early stage in the develop-

ment I feel that at best it is a clumsy tool. Therefore, we should be extremely careful about raising false hopes.

On the other hand, the very complete presentation of Donn Parker shows a tremendous effort in this direction. And whatever the relative efficiencies of this tool, I "drool" when thinking about the possibility of using the system. In fact, I wonder about incentives to minimize the "playing around" of the operator?

LANGUAGES

As illustrated in the MAC system and in Donn Parker's system, we are going to see many problem oriented language systems for use with remote stations. So far, many of these are adaptations of batch processing languages, but time will see the development of systems optimized for remote-on-line use.

In the area of procedure oriented languages QUIKTRAN is an adaptation of the batch processing language FORTRAN. In reviewing Johnson's PAT language one wonders if it could not have been developed with a much closer relationship to either ALGOL or NPL. For language exploration this is not too important. However, if anyone is developing a language which he expects to be widely used, then he should base it on either ALGOL or NPL if possible.

SUMMARY

I am impressed by the large show of interest in the subject, and I hope this is symptomatic of a hard look at the problems of improving batch processing systems and, more importantly, at the greater variety of tools available by improved man-machine interface equipment (hardware and software). I feel that the culmination of the developments described in this set of papers marks the first real step in improving the use of the computer as a research and development tool. In other words, making the computer available to an individual on a more or less continuous basis is the first significant step in improved use of computing equipment since the first automatic computer was put into operation in 1949.

ATTENDEES

List of Attendees
Symposium on On-Line Computing Systems

NAME	AFFILIATION
Robert Abbott	*Lawrence Rad. Lab.*
Milton B. Adams	*Stanford Res. Inst.*
Gale R. Aguilar	
K. D. Allen	*Beckman Instruments*
Robert H. Allen	*Moore Business Forms*
James R. Ameling	*Technical Operations*
Jeffry Amsbaugh	*Sun Oil*
Martin Anderson	
Sypko Andreae	*Lawrence Rad. Lab.*
Frank B. Andrews, Jr.	*NCR Company*
G. W. Armerding	*RAND Corp.*
W. D. Armour	*IBM*
Wendell Arntzen	*Boeing Company*
Lester G. Arnold	*Eastman Kodak Co.*
James Ashbaugh	*UNIVAC*
J. R. Ashcraft	*Sandia Corp.*
H. L. Asser	*IBM*
Marc Auslander	*IBM-Boston*
Herbert F. Ayres	*Morgan Guaranty Trust*
C. L. Baker	*RAND Corp.*
Frank R. Baldwin	*IBM*
Berton E. Barker	*Sandia Corp.*
Hugo H. Barlow, Jr.	*Aerojet-General Corp.*
Sheridan F. Barre	*Natl. Cash Register Co.*
A. Lewis Bastian, Jr.	*IBM*
K. E. Batcher	*Goodyear Aerospace Corp.*
Edgar A. Bates	*Stromberg Carlson Corp.*
Robert S. Bauder	*Security 1st Natl. Bank*
Fred W. Bauer	*Burroughs Corp.*
Bob Bearden	
George C. Beason	*Natl. Cash Register Co.*
T. J. Beatson	*General Electric Co.*
Dr. D. C. Beaumariage	*Bunker-Ramo Corp.*
Theodore S. Beck	*UNIVAC*
John D. Beierle	
Alan Bell	
Robert W. Bemer	*UNIVAC*
J. Philip Benkard	*IBM*
J. Russ Bennett	*Burroughs Corp.*
James Bennett	
John R. Bennett	*IBM*
Evelyn Berezin	*Digitronics Corp.*
James O. Berish	*IBM*
Marvin Berlin	*Gen. Instrument Corp.*
Martin F. Berman	*Natl. Cash Register Co.*
U. Berman	*IBM*
Robert Bernard	*Perkin-Elmer Corp.*
P. Berning	*TRW Space Tech. Lab.*
Marvin N. Bernstein	*Litton Industries*
Mort Bernstein	
Hilmer W. Besel	*La Sierra College*
L. P. Best	*Lockheed Calif. Co.*
Thomas P. Bianco	*IBM*
Lyle P. Bickley	*IBM*
Eugene P. Binnall	*Lawrence Rad. Lab.*
Dr. Garrit A. Blaauw	*IBM Data Systems Div.*
Max Blakely	*Boeing Co.*
Hawley Blanchard	*Planning Res. Corp.*
R. J. Blanken	*Bunker-Ramo Corp.*
Martha Bleier	*System Dev. Corp.*
R. E. Bleier	*System Dev. Corp.*
Melvin H. Blitz	*Sylvania Elec. Systems*
Joseph Blum	
Brooke W. Boering	*Talman Federal Savings*
Joe Bogar	*Friden, Inc.*
Robert G. Bolman	*Bunker-Ramo Corp.*
Robert Bomeisler	
Elaine R. Bond	*IBM-Boston*
Richard C. Bond	
R. U. Bonnar	*Shell Development*
J. M. Bookston	*General Motors*
J. C. Borgstrom	*UNIVAC*
A. A. Borsei	*Beckman Instruments, Inc.*
J. F. Bost	*Electro-Mech. Res., Inc.*
Edward C. Boycks	*Raytheon Computer*
Douglas N. Brainard	*Lawrence Rad. Lab.*
Daniel Brayton	*Sanders Assoc., Inc.*
W. J. Brian	*Control Data Corp.*

Harold L. Brint	*Sandia Corp.*
Wm. Broderick	*General Electric Co.*
Charles Brodnax	*Texas Instruments, Inc.*
Robert J. Broen	
Charles Broman	*White Sands Mis. Range*
J. H. Brown	*Mitre Corp.*
J. Reese Brown, Jr.	*Burroughs Corp.*
Robert J. Brown	*Natl. Cash Register Co.*
Ross H. Brown	*Univ. of Calif.*
Amcel Buchanan	*Tinker AFB*
Capt. Thomas K. Burgess	*USAF Office of Sci. Res.*
David L. Bussard	*Hughes-Fullerton*
E. Daniel Butler	*Ernst and Ernst*
E. D. Callender	*Aerospace Corp.*
Carlo Paul Calo	*USN Sta., Bl. 196, Washington*
S. H. Cameron	*I. I. T.*
Richard G. Canning	*Canning Publications, Inc.*
F. C. Carlin	*Lockheed Calif. Co.*
Charles E. Carlson	*N. Y. St. Dept. Public Works*
Charles H. Carman	*U. S. Dept. of Interior*
J. E. Carrico	*Union Bk. Comp. & Serv. Ctr.*
Keith Carter	*USAF*
Arthur F. Casey	*Stromberg Carlson*
C. T. Casale	*Control Data Corp.*
Perry Cassimus	*Natl. Cash Register Co.*
David Caulkins	*Caulkins Assoc.*
James V. Cellini	*USAF*
Herb Chalmers	*ITT-DISD*
Carl Chamberlin	*IBM*
S. H. Chasen	*Lockheed Georgia Co.*
Tien Chi Chen	*IBM*
Leonard G. Chesler	*RAND Corp.*
Clayton Chisum	
Duane M. Christ	*IBM*
Lloyd Christianson	
George J. Christy	*State of Calif.-DWR*
D. L. Clark	*IBM*
Charles A. Clark	*MIT Lincoln Lab.*
David L. Clark	*Bell Telephone*
Donald J. Clark	*United Res. Serv. Corp.*
Dorothea S. Clarke	*General Electric Co.*
Donald M. Clarry	*Technical Operations*
Daniel P. Clayton	*Bell Telephone Labs.*
Richard Cleaveland	*U.S. Govt.*
Wm. Clelland III	*USN Ord. Test Sta.*
C. T. Clingen	*General Electric Co.*
Richard Clippinger	*Honeywell Inc.*
W. R. Coates	*AT&T Co.*
E. G. Coffman	*System Dev. Corp.*
Victor Colburn	
R. Wade Cole	*Stanford University*
B. D. Collins	*McDonnell Automation Ctr.*
Donald Colvin	
Boyd F. Connell	*Owens-Illinois*
Jack Connolly	*Holiday Inns of America*
Herbert F. Congram	*General Electric Co.*
George E. Comstock	*Friden, Inc.*
Carl J. Conti	*IBM Dev. Lab.*
Robert P. Cook	*US Army Eng. Div., So. Pac.*
Charles Corderman	*Scientific Eng. Inst.*
E. A. Corl	*R.C.A.*
Charles F. Corley	*IBM*
W. L. Coultas	*Shell Oil Co.*
Daniel L. Covill	*Burroughs Corp.*
Daniel E. Cowgill	*Research Analysis Corp.*
E. C. Craddock	*Aerojet-General Corp.*
Lt/Col. Albert Crawford	
Charles Crawshaw	*General Electric Co.*
James J. Crockett	*USN*
Timothy Cronin	*USMC*
F. C. Cunningham	
Wm. S. Currie	*Texaco, Inc.*
D. J. Dantine	*Clark Equipment Co.*
Howard Davis	*USAF*
George W. Dawson	*IBM*
R. L. Day	*Natl. Cash Register Co.*
George DeFlorio	*System Dev. Corp.*
James W. Dening	*Dept. Hlth. Educ. & Welfare*
Donald E. Denk	*Pacific Missile Range*
R. P. D'Evelyn	*Autonetics, Div. N. Am. Av.*
Richard Devereaux	*IBM*
Robert J. Dolan	*Clerk, Bd. Supervisors*
John P. Dolch	*University of Iowa*
Fletcher Donaldson	
Terence P. Donohue	*Control Data Corp.*
Richard Dooley	*1st Natl. Bk. of Chicago*
Richard Dorrance	*United Res. Serv. Corp.*
Roger R. Dougan	*Westinghouse Elec. Corp.*
James O. Drake	*Arthur D. Little, Inc.*
Walter F. Dudziak	*General Electric Co.*
Douglas M. Duffy	*C. I. A.*
Theodore M. Dunn	
D. B. Earl	*Control Data Corp.*
Tom S. Eason	*Mesa Scientific Corp.*
Patricia Eddy	*System Dev. Corp.*
H. P. Edmundson	
Charles M. Edwards	*Bendix Corp.*
R. E. Edwards	*General Electric Co.*
John B. Eichler	*ITT Research Inst.*
Vernon Eisenbraun	
J. Eliades	*Aerospace Corp.*
B. Elliot	*TRW Space Tech. Lab.*

R. Ellison	*U. S. Steel*
Herman Englander	*US Navy Electronics Lab.*
William English	
Barry Epstein	
Fred Erman	
Nathan Estersohn	*Bendix-Pacific Div.*
R. E. Etheridge	*General Electric Co.*
Bob O. Evans	*IBM*
George F. Fagan	*The MITRE Corp.*
H. M. Farmer	*Aerospace Corp.*
Wm. A. Farrand	*Autonetics*
C. C. Farrington	*TRW Space Tech. Labs.*
Alex G. Fedoroff	*Grumman & Assoc., Inc.*
C. Nick Felfe	*Bonner & Moore Assoc.*
B. Fenton	*I. C. I. (New York) Inc.*
Harold D. Feldman	*UNIVAC*
Marion L. Fickett	*Inst. for Def. Analyses*
Boris Field	*NCR-ED*
Robert C. Fife	*UNIVAC*
Sidney I. Firstman	*RAND Corp.*
Donald D. Fisher	*Stanford University*
F. P. Fisher	*IBM*
A. M. Fleishman	*R. C. A.*
George W. Fleming	*Eastman Kodak Co.*
Merrill M. Flood	*Univ. of Michigan*
Walter W. Flood	*Matrix Corp.*
Larry Forrest	*White Sands Mis. Range*
Ed Forsberg	*Continental Oil Co.*
Wm. Fortenberry	*NASA-Marshall*
Carl Forth	*Lockheed Calif. Co.*
K. Scott Foster	*Honeywell*
R. S. Fox	*General Electric Co.*
S. J. Fox	
James W. Fraher	*United Air Lines*
Walter Fredrickson	*Radiation, Inc.*
J. A. French	*Natl. Cash Register Co.*
W. Fred Frey	
Dr. H. Friedman	
W. H. Frye	*US Navy Electronics Lab.*
W. H. Fuhr	*Melpac Inc.*
Heinz Gabloffsky	*No. Amer. Avia. S&ID*
L. E. Gallaher	*MTS Bell Tel. Labs.*
Lewis Gallenson	*System Dev. Corp.*
Richard J. Garcia	*Rexall Drug Co.*
J. E. Garvin	*UNIVAC*
Alan E. Geiger	*Space Tech. Lab.*
H. Gelernter	*IBM Research*
C. L. Gerberich	*General Electric Co.*
Herbert Getreu	
William B. Gibson	*IBM*
Kenneth R. Gielow	*Lockheed*
R. Gillespie	*Control Data Corp.*
John T. Gilmore	
Robert Gilmour	*Stanford Research Inst.*
R. G. Glaser	*McKinsey & Co., Inc.*
Edward Golden	
Phillip Goldstein	*Rexall Drug Co.*
D. R. Goodchild	*Imperial Chem. Ind., Ltd.*
Harry S. Gopies	*Control Data Corp.*
Harold Gottheim	*N. Y. St. Dept. Public Works*
Andrew M. Gould	*Friden, Inc.*
Kent Gould	*Douglas Aircraft MSSD*
Charles Grayson	*Tulane University*
P. A. Greco	*Kellogg Company*
Dale Green	*Electro-Optical Systems*
T. H. Green, Jr.	*Shell Oil Co.*
Leonard Greenberg	
T. S. Greenwood	*Bell Telephone Labs.*
E. Griesheimer	*Douglas Aircraft Co.*
Harold Paul Gross	
Herbert L. Gross	*Natl. Cash Register Co.*
Harry Grossman	*Security 1st Natl. Bank*
Robert Grummett	*Natl. Cash Register Co.*
James R. Guard	*Applied Logic Corp.*
Robert Gunderson	*Natl. Cash Register Co.*
Frank Gummersall	
Lothar F. Haas, Jr.	*Natl. Cash Register Co.*
Wm. Hagerbaumer	*Electronic Assoc. Inc.*
Vern E. Hakola	*Touche, Ross, Bailey, Smart*
Norton P. Halber	
A. Halenbeck	*Aerospace Corp.*
L. B. Hamilton, Jr.	
Iver C. Hansen	*Burroughs Electro-Data*
Warren B. Harding	*IBM*
D. L. Harmer	*Autonetics, Div. N. Amer. Av.*
J. Harrington	*City & County, San Francisco*
Dr. J. Harriman	*General Electric Co.*
Don F. Harroff	*Research Labs, GMC*
H. P. Hart	*Hughes Aircraft*
A. H. Hassan	*Natl. Cash Register Co.*
Thomas E. Hassing	
Dean A. Hatfield	
Erwin A. Hauck	*Burroughs Electro-Data*
J. D. Hawkins	*Lockheed Calif. Co.*
Clark Hayes	
Robert F. Hays	*Bunker-Ramo Corp.*
Halroyd Haywood	*Security 1st Natl. Bank*
Stanley Hawkinson	
George Hazelworth	
James L. Heard	*No. Amer. Aviation*
Frank R. Heath	*Westinghouse Elec. Corp.*
R. C. Heath	*IBM*

W. J. Heffner	General Electric Co.
L. B. Heggie	
Armin W. Helz	US Geological Survey
Prof. Hetherington	University of Kansas
Bill Herr	
E. D. Hildreth	NASA
Munson Hinman	
Verlin Hoberecht	IBM
Donald Hodges	Argonne Natl. Lab.
G. E. Hoernes	IBM
D. J. Haflinger	Stromberg-Carlson
Robert R. Hohl	Naval Res. Lab.
Dean A. Holdiman	
Thomas Holloran	Natl. Cash Register Co.
Fred Holloway	Lawrence Rad. Lab.
John Holloway	Douglas Aircraft Co.
Robert Hollitch	IIT Research Inst.
Samuel Hoover	IBM
Wm. J. Hoskins	Douglas Aircraft Co.
Donald Houck	United Res. Serv. Corp.
Richard Hovey	Booz-Allan Applied Res.
R. C. Howard	Giannini Controls Corp.
Robert Hsu	Natl. Bureau of Stdrds.
Arnold H. Hubert	Computer Usage Co.
Willis Hudson	Control Data Corp.
Larry D. Hughes	Computer Applications
D. Hull	Chrysler Corp.
G. O. Hummel	No. Amer. Aviation
Roger L. Hummel	Rexall Drug Co.
Frank J. Hundt	Natl. Cash Register Co.
Norman L. Hunt	IBM
Bernard Hurley	RCA
Daniel T. Hurley	
Pauline Hutter	
Jack G. Ianotti	
F. Ivie	TRW Space Tech. Lab.
Thomas Jackson	Southwest Res. Inst.
S. L. Jamison	
Lt/Col. W. Jarrell	USAF Academy
E. A. Jennings	UNIVAC
M. K. John	U. S. Steel Corporation
N. E. Johnson	Lockheed Calif. Co.
Isabel F. Johnston	
Robert Johnston	
Gary W. Jones	Burroughs Corp.
Glyn H. Jones	Burroughs Corp.
Frank G. Jordan	IBM
Earl C. Joseph	UNIVAC
K. R. Joseph	NASA
Theodore Kallner	IBM
Stanley L. Kameny	System Dev. Corp.
H. Kanner	Control Data Corp.
J. D. Kassan	Western Union Tel. Co.
David Katch	No. Amer. Aviation
Dr. Julian Kately	Michigan State Univ.
Lt. H. Kaufman	USAF
Arthur Kaupe, Jr.	Westinghouse Res. Lab.
Bob Kawaguchi	L.A. Dept. W & P
C. S. Kazek, Jr.	Los Alamos Scien. Lab.
Roy Keir	Beckman Instruments
J. P. Kelly	IBM
Mary Anne Kelley	Brookhaven Natl. Lab.
Neal M. Kendall	Natl. Cash Register Co.
S. V. Khoury	MTS—Bell Labs.
Douglas T. Kielty	Burroughs Electro-Data
Daniel B. Killeen	Tulane University
F. J. Killian	Northrop Corp.
G. W. Kimble	TRW Space Tech. Labs.
Paul D. King	UNIVAC
H. A. Kinslow	IBM
Elliot B. Kleiman	Martin
John R. Knight	IBM
Fred Koepping	UNIVAC
George M. Kohn	IBM
Alexander Korwek	Bacchus Works
W. J. Kosinski	General Electric
Roger P. Kovach	Naval Supply Center
Dr. M. Krichevsky	NIDR, NIH
Dean C. Kriebel	Sun Oil
Norman L. Krueder	Burroughs Electro-Data
Gene Kucinkas	Foxboro Co.
Akio Kumamoto	Data Processing
Francis Kurriss	Douglas Aircraft MSSD
J. M. Kusmiss	IBM
Wm. J. LaBelle	Security 1st Natl. Bank
R. P. Lagerstrom	Stanford University
John A. Lamkin	Datatrol Corp.
Charles Lander	NASA
Gerald Landsman	Motorola
G. L. Lane	Sandia Corp.
Robert E. Laws	U.S. Steel Corp.
Bennett S. Lebow	DA, DCSLOG Data Proc. Div.
Sidney Lechter	NASA
David Legge	IBM
Ronald P. Leinius	Union Carbide
Leon Leskowitz	U.S. Army Elec. Labs.
Paul Leslie	Westinghouse Electric
Donald L. Lewis	IBM
Frank J. Lewis	Radiation, Inc.
Neil Lewis	
S. H. Lewis	Aerospace Corp.

Arlyn G. Liddell	*Boeing Co.*
D. C. Lincicome	*Stanford Research Services*
Neil Lincoln	*United Research Services*
Andrew T. Ling	
Peter Linz	*IBM*
G. J. Liviakis	*Beckman Instruments, Inc.*
Richard Loewe	*Aeronutronic Div.*
Edward Loges	*Booz-Allan Applied Res.*
David H. Long	*USAF*
Paul J. Long	*Holiday Inns of America*
E. B. Loop	*Union Oil Co. of Calif.*
Floyd Looschen	*Burroughs Electro-Data*
Jack A. Lord	*Gen. Tel. Co. of Calif.*
John F. Lubin	*University of Pa.*
V. H. Lucke	*General Electric Co.*
Donald Lumbard	*Natl. Cash Register Co.*
Dr. R. F. Lyjak	*University of Michigan*
H. Lykken	*Minn.-Honeywell Reg. Co.*
Hugh J. Lynch	*Natl. Cash Register Co.*
John C. Lynn	*NASA*
J. D. Lyon	*Sandia Corp.*
Alan C. MacDonald	*IBM*
Roger A. MacGowan	
A. Maclean	*Minn.-Honeywell Reg. Co.*
Wellen B. MacLean	*Union Carbide Corp.*
John B. MacLeod	*NASA*
Patrick McGovern	*USN*
Thos. McKinney	*USNOTS*
Harold Malliot	*Stanford Research Inst.*
Donald Machen	*Lawrence Rad. Lab.*
Charles Mackenzie	*IBM*
F. B. Mackenzie	*Burroughs Corp.*
Dr. Dale Madden	*IBM*
J. L. Maddox	
R. J. Maguire	*Automatic Elec. Co.*
R. A. Mallet	*Autonetics, No. Amer. Av.*
Allan Mandelin	
E. L. Manderfield	
Wilbur Mann	*United Aircraft Corp.*
John J. Manyak	
R. B. Mapes	*Douglas Aircraft Co.*
Wm. Marchman	*IBM*
John Marez	*U.S. Navy Electronics Lab.*
Lowell G. Market	*UNIVAC*
N. A. Martellotto	*Bell Telephone Co.*
Milton Martenson	*Dept. of Defense*
Norman Marshall	*Bendix-Pacific Div.*
Maj. Wm. Marsland	*Seiler Research Lab.*
Samuel Matsa	*IBM*
Jesse Maury	*NASA—Greenbelt*
Justin N. Mead	*Dow Chemical*
M. D. Mayer	*Douglas Aircraft Co.*
David F. McAvinn	*Foxboro Co.*
J. S. McBirney	*Shell Oil*
Patrick McClung	
C. C. McClurkin	*Bonner and Moore Assoc.*
George McCormick	*Aerojet-General Corp.*
Gerald McFadden	*IBM*
L. E. McHenry	
C. R. McKelvey	*Sandia Corp.*
John B. McLean	*Rome Air Dev. Center*
Harvey McMains	*Amer. Tel. & Tel. Co.*
Claude McMillan	*Michigan State University*
Torben Meisling	*Stanford Research Inst.*
David Meldrum	*United Res. Serv. Corp.*
W. R. Melton	*Inland Steel Co.*
Sir J. S. Menteth	*Imperial Chem. Ind., Ltd.*
Don W. Mercer	*Douglas Aircraft Co.*
John F. Meyer	*University of Michigan*
Karl Meyer	*No. Amer. Aviation*
Bart Michielsen	*Natl. Cash Register Co.*
Wren Middlebrook	
Barrett Miller	*Naval Supply Center*
John F. Miller	*IBM*
Stephen Miller	*Stanford Research Inst.*
Warren Milroy	*U.S. Navy Electronics Lab.*
Lester Mintzer	*Interstate Elec. Corp.*
Ronald Miranda	*Tech. Operations Res.*
James Misho	
Charles Missler	*Ford Motor*
Don E. Mitchell	*AT&T*
John Mitchell	*Stanford Research Inst.*
Rick M. Moore	*United Res. Serv. Corp.*
John C. Morgan	*Bunker-Ramo Corp.*
Maynard Morris	*IBM*
J. P. Morrison	*IBM*
Arthur Moskin	*U.S. Navy Dept.*
Kern A. Moulton	*J. P. Stevens & Co., Inc.*
J. R. Mueller	*IBM*
George Mulholland	*Assoc. Universities, Inc.*
Joseph P. Murphy	*IBM*
Wm. Murray	*Tech. Operations, Inc.*
Henry F. Nanjo	
S. A. Nastro	*IBM*
Michael R. Nekara	*IBM*
J. O. Neuhaus	*Control Data Corp.*
R. A. Nichols	*Autonetics, N. Amer. Av.*
Ralph Niemi	*U.S. Navy Elec. Lab.*
W. E. Niemond	*General Precision*
Albert Noble, Jr.	
Jerre D. Noe	*Stanford Research*
R. V. Norvill	*Sandia Corp.*

R. W. Notto	*UNIVAC*
Thomas Nourse	*Foxboro Co.*
Wm. H. Ninke	*Bell Telephone Labs.*
Clark Oliphint	*Burroughs Electro-Data*
Glenn A. Oliver	
Ron Olsen	*United Research Services*
Joseph Olsztyn	*General Motors*
James J. O'Neill	
H. N. Oppenheimer	*CBS Labs.*
Robert C. Oram	*IBM*
James O'Reilly	*Assoc. Universities, Inc.*
Dr. R. Orensteen	*IBM*
Vance E. Page	*Pacific Tel. and Tel.*
D. J. Pagelkopf	*Control Data Corp.*
Maxwell O. Paley	*IBM*
Oscar M. Palos	
Alexander Papp	*Taylor Township*
Gabriel A. Pall	*IBM*
N. C. Panella	
R. A. Paris	*Lockheed*
Wm. W. Parker	*Burroughs Corp.*
R. Pash	*Minneapolis-Honeywell*
Frank Patton	*Lawrence Radiation Lab.*
P. F. Paul, Jr.	*Autonetics, N. Amer. Avia.*
C. R. Pearson	*J. P. Stevens & Co., Inc.*
Ray S. Peck	*IBM*
Gordon Pelton	*Lockheed Missiles*
John Pendleton	*Douglas Aircraft MSSD*
James W. Perry	*University of Arizona*
W. M. Perry	*IBM*
Eugene A. Peters	*No. Amer. Aviation*
Kenneth Pergande	*Natl. Cash Register Co.*
Bernard Peters	
Walter N. Phillips	*Bunker-Ramo Corp.*
Wm. S. Pickrell	*Mandrel Ind. Inc.*
P. E. Piechocki	*Richfield Oil Corp.*
Charles L. Pierce	*Stanford Research Inst.*
M. A. Pilla	*Bell Telephone*
Thomas G. Pine	*Digital Dev. Corp.*
Elliot Pinson	*Bell Telephone Lab.*
B. W. Pogorelsky	*Natl. Cash Register Co.*
Steven T. Polyak	*Inst. for Def. Analyses*
A. G. Pontius	*IBM*
Ted Poole	*Scantlin Elec., Inc.*
Ivan J. Popejoy	*General Electric Co.*
Fred Pradko	*U.S. Army*
L. I. Press	
Eugene R. Puckett	
Thomas M. Putnam	*University of Calif.*
P. H. Pyle	*Tidewater Oil Co.*
F. I. Quinlan	*Douglas Aircraft Co.*
Peter M. Rakich	*So. Permanente Serv., Inc.*
R. W. Ralston	*Amer. Tel. and Tel. Co.*
Walter Ramshaw	*United Aircraft Corp.*
Russell W. Ranshaw	*University of Pittsburgh*
Wayne F. Rayfield	*University of Wisconsin*
Donald E. Rea	*General Dynamics*
Daryl Reagan	*Stanford University*
Ottis W. Rechard	*N.S.F.*
Samuel C. Reese	*Honeywell, Inc.*
Otto Reichardt	*Raytheon Computer*
A. Reichenthal	*Hughes Aircraft*
Roger L. Reifel	*General Dynamics/Astro.*
Harry Reinstein	*IBM*
Dr. W. Rheinboldt	*University of Maryland*
V. Thomas Rhyne	*NASA—Langley*
Dr. C. N. Rice	*Eli Lilly & Co.*
John E. Rideout	*IBM*
A. O. Ridgway	*IBM*
Thomas L. Ringer	*IBM*
John F. Riordan	*University of Michigan*
Dan Robbins	*Benson Lehner*
Lt. K. Robbins	
C. R. Roberts	*No. Amer. Aviation*
W. C. Rockefeller	
C. E. Rodemann	*IBM*
LeRoy J. Rodgers	*No. Amer. Aviation*
George F. Roe	*The Boeing Co.*
C. N. Rollinger	*Whirlpool Res. Labs.*
W. E. Rood	*U.S. Steel*
Henrik M. Roos	*Naval Supply Center*
Herb Rosenheck	*Hoffman Electronics*
Arthur V. Rubino	*Advanced Scientific Inst.*
Harvey Rubinstein	
Joseph Rue	*No. Amer. Avia.*
Linus F. Ruffing	*C.I.A.*
Arnold Ruskin	*Harvey Mudd College*
Richard Russell	*IBM*
R. J. Ruud	*IBM*
Joseph Ryan	*USAF*
J. J. Salyers	*Natl. Cash Register Co.*
Harold Sackman	*System Dev. Corp.*
E. J. Samuelli	*Kennecott Copper Corp.*
Lt. Col. H. Sanders	*USAF*
Margo A. Sass	*Off. of Naval Res.*
Dr. H. Sassenfeld	*General Electric Co.*
Kirk Sattley	*Computer Associates, Inc.*
Marshall Savage	*Universal City Studios*
Don Savitt	*Hughes Aircraft*
David Saylor	
Ronald Schauer	*Automatic Electric Labs.*
Jerold Schleicher	*Union Bank*

R. G. Schluter	*Space Technology Labs.*
George Schmidt	*USAF*
L. A. Schmittroth	*MA & MC PPCO*
Robert Schnuck	*IBM*
Vernon D. Schrag	*IBM*
Donald P. Schultze	*N.S. Navy Dept.*
Dave Schumacher	*USAF*
G. P. Schumacher	*U.S. Navy Elec. Lab.*
Betsy Schumacker	*IBM*
M. H. Schwartz	*Federal Reserve System*
Robert Schwartz	*Raytheon Company*
Walter Schwartz	*MITRE Corp.*
V. J. Scimone	*IBM*
Sheldon Simmons	*Electronic Specialty Co.*
Harry Schwartz	
Dan W. Scott	*General Electric Co.*
Richard B. Scott	*General Electric Co.*
J. Sedgewick	
Wm. S. Selers	*Sears Roebuck*
Wm. A. Sensiba	*Western Electric Co.*
Lt. E. A. Schmidt	*USAF*
A. M. Shalloway	*Natl. Radio Astr. Obs.*
Wm. C. Shannon	*Crocker-Citizens Natl. Bk.*
Theodore Shapin	*Beckman Instruments*
Richard S. Sharp	*Burroughs Electro-Data*
Christopher Shaw	*System Dev. Corp.*
J. C. Shaw	*RAND Corp.*
Regis Shinger	
James L. Shira	*United Aircraft Corp.*
H. L. Shoemaker	*Bunker-Ramo Corp.*
Glenn W. Shook	*Dept. of Defense*
Al Shore	*Burroughs Corp.*
W. L. Sibley	*RAND Corp.*
H. W. Siegmann	
Lt. R. Singleton	
C. E. Skidmore	*Westinghouse Electric*
N. J. Skoner	*Clark Equipment Co.*
Robert Skoog	*Lawrence Radiation Lab.*
Derrel L. Slais	*Control Data Corp.*
Carl O. Smarling	*IBM*
Blanchard Smith	*Melpar, Inc.*
Earle Smith	*Natl. Cash Register Co.*
R. J. Smith	*Control Data Corp.*
R. D. Smith	*R.C.A.*
Robert W. Smith	*IBM*
R. L. Snyder	*Western Union Tel. Co.*
Glenn D. Sorensen	*Honeywell E.D.P.*
Branko Soucek	*Brookhaven Natl. Lab.*
Arthur Speckhard	*Bellcomm, Inc.*
Monroe Spierer	*Technical Operations, Inc.*
Robert J. Spinrad	*Brookhaven Natl. Lab.*
James Spitze	*Friden Inc.*
Donald R. Spivey	*IBM*
Conrad R. Springer	*IBM*
Jon S. Squire	*Westinghouse Elec. Corp.*
Howard R. Stagner	
Richard H. Stark	*Washington State Univ.*
Earl Stearns	
Lee Stephens	*Friden, Inc.*
F. G. Stockton	*Shell Dev. Co.*
David R. Stolz	*American Tel. and Tel.*
D. R. Strickland	
Donald M. Styer	*Union Carbide Corp.*
George R. Sugar	*Natl. Bureau of Stdrds.*
R. J. Sullivan	*General Motors Res. Lab.*
Audrey Summers	*NASA*
John Sutherland	*IBM*
Dr. G. H. Swift	*IBM*
Bruce Sypkens	*Dept. of Defense*
Jeffrey Tai	*Natl. Cash Register Co.*
G. Platt Talcott	*Raytheon Co.*
Robert G. Tantzen	
Mary Tate	*USAEL, Ft. Monmouth, N. J.*
Joseph E. Taylor	*Honeywell, Inc.*
Norman H. Taylor	*Control Data Corp.*
Richard Tedrick	*NASA*
Kas Terhorst	*Burroughs Corp.*
Thomas Theberge	*Douglas Aircraft Co.*
William Thelsner	*Control Data Corp.*
Don Thiede	*IBM*
David J. Thomas	*MITRE*
G. A. Thompson	*Honeywell, Inc.*
K. L. Thompson	*State Govt.*
R. Tink	*Natl. Cash Register Co.*
N. Ralph Tipaldi	*Perkin-Elmer Corp.*
Dale Tolin	*Phillips Petroleum Co.*
Dolan H. Toth	*Control Data*
R. E. Trainer	*Nortronics*
A. J. Trangle	*UNIVAC-Div. Sperry Rand*
N. E. Truitt	*Shell Dev. Co.*
H. S. Tsou	
J. C. Tu	*General Precision, Inc.*
S. T. Tuttle	*Kellogg Co.*
Donald L. Trask	*Lockheed Mis. & Space Co.*
Fred Tremaine	*County of Sacramento*
Rein Turn	*RAND Corp.*
H. F. Tweeden	*IBM*
Gordon T. Uber	*Lockheed*
B. A. Udovin	*Bunker-Ramo Corp.*
R. H. Udy	*Aerojet-General Corp.*
L. L. Van Oosten	*Allstate Ins. Co.*
R. Van Buelow	*System Dev. Corp.*

T. A. Van Wormer	*Univ. of Pittsburgh*
E. R. Vance	*General Electric Co.*
Howard Vann	*USAF*
R. S. Vatilla	*Goodyear Aerospace Corp.*
A. H. Vorhaus	*System Dev. Corp.*
Paul L. Voss	*Navy Electronics Lab.*
J. W. Wager	*Hughes Aircraft Co.*
R. A. Wagner	*RAND Corp.*
F. W. Wallace	*Citrus College*
John Walley	*Hughes-Fullerton*
W. C. Walter	*Northrop Corp.*
B. P. Warner	*Edgerton, Germeshausen & Grier*
E. Wasserman	*IBM*
Gordon M. Watson	*Computer Usage Co.*
Paul V. Webb	*Phillips Petroleum Co.*
Harlan Webster	*Douglas Aircraft Co.*
Gary L. Weeks	*Kennecott Cooper Corp.*
J. H. Wegstein	*Natl. Bureau of Stds.*
Dr. B. Weiss	*Johns Hopkins University*
Eric A. Weiss	*Sun Oil Co.*
R. C. Wheeler	*Airborne Instr. Lab.*
William Whitacre	
Dr. Oliver Whitby	*Stanford Research Inst.*
Carl L. White	*Natl. Inst. of Health*
Richard A. White	*Inst. for Def. Analyses*
W. Y. Whittemore	*General Electric Co.*
H. R. Widmann	*General Tel. Co.*
J. B. Wiener	*General Electric Co.*
Richard B. Wilke	*IBM*
W. Wilkinson	*Bunker-Ramo Corp.*
C. A. Williams	*Collins Radio Co.*
E. R. Williams	*IBM*
Mary Williamson	*General Electric Co.*
Robert Williams	
Raymond Wilser	*IBM*
R. C. Wilson	*University of Michigan*
William Wilson	*Cutter Lab.*
James E. Wollum	*Burroughs Electro-Data*
Vincent Wollum	*Archer-Daniels-Midland Co.*
Pearson L. Wood	*IBM*
H. D. Woods	*H. D. Woods & Associates*
Thomas F. Woods	*NASA—Houston*
W. E. Woods	*Computer Control Co.*
L. H. Woodward	
Stanley H. Woster	*Space Tech. Labs.*
William Wright	*General Electric Co.*
H. Wyle	*Autonetics, Div. N. Am. Avia.*
Tak Yamashita	*Bunker-Ramo Corp.*
Georgiana Yang	*IBM*
Norman Yarosh	
Thomas L. Yates	*Oregon State Univ.*
Robert C. Yens	*MITRE*
A. W. Yonda	*RCA*
M. G. Young	*Shell Dev. Co.*
V. J. Zapotocky	*General Electric Co.*
D. C. Zatyko	*SPO*
James R. Ziegler	*Natl. Cash Register Co.*
Charles A. Zuroff	*Brookhaven Natl. Lab.*