ON-LINE COMMAND AND CONTROL STUDY

D58-5U1

W. D. Wilkinson and G. Martins

PART I

Annual Progress Report

September 1965

Prepared for

Office of Naval Research

Information Systems Branch

Contract Nonr. 4182(00)

THE BUNKER-RAMO CORPORATION

DEFENSE SYSTEMS DIVISION

8433 FALLBROOK AVENUE • CANOGA PARK CALIFORNIA 91304

ABSTRACT

In most automated Command Information Systems, the military user is hampered by the lack of convenient means for communication with the computer system and by the inflexibility of the system to respond to a rapid change either in the type of problem to be solved or in the method of solving a standard problem. It is in these areas that on-line techniques show promise for improvement. The major objective of on-line techniques is to put the user directly in the problem-solving system loop in such a manner that the experience and intuition of the user becomes closely coupled with the powerful computational capabilities of the computer so that there is a balanced interaction between the capabilities of each.

On-Line techniques employ a functional approach to permit the user to:

1) communicate with the computer system conveniently in his own terminology,
2) structure the problem solving process, and
3) maintain continuous control over the system.

Although Command and Control is emphasized in this study, the techniques described are general and should be useful in a wide range of problem solving applications.

ACKNOWLEDGMENT

This research is supported by the Office of Naval Research under Contract Nonr-4182(00).

The advice and assistance of Dr. W. Watson and Dr. G. Culler is acknowledged. Important contributions have been made by Jim Weaver and Tom Tack.

Naval personnel from CinCLANT, Cinc PAC, and NEL have been very helpful in providing advice and constructive criticism in the Naval operational areas. The opinions stated are those of the authors.

TABLE OF CONTENTS

		Page
Abstract		-ii-
Acknowledgement		-iii-
List of Illustrations		-v-
I.	Overview	1.
II.	Introduction	4.
III.	On-Line Systems	6.
	General	6.
	Objectives	6.
	B-R On-Line Techniques	9.
IV.	Application of On-Line to Command and Control	14.
	Command Information Systems	14.
	On-Line Experimental System	19.
V.	Bibligraphy	34.
Appendix	Basic Programming System	A1

LIST OF ILLUSTRATIONS

Figure		Page
1.	Conventional System Structure	18
2.	On-Line System Structure	20
3.	Bunker-Ramo 85 Display/Control Console	22
4.	Overlay Keyboard	23
5.	Data Base	27
6.	Nouns	28
7.	Verbs	30

I. <u>Overview</u>

Contract Nonr 4182(00) was undertaken to investigate the feasibility of applying on-line techniques to Navy Command and Control problems. The problem area selected was that of a specified command such as CinCLANT. No attempt was made to simulate a specific operation but rather the general problem of automating the Command Information function was investigated to determine if On-Line techniques would be applicable.

It was determined that On-Line techniques did indeed show promise in overcoming some of the problems.

In June of 1964, a demonstration was given of an initial limited system. Basically, the demonstration consisted of three elements:

(1) The data base, including area map.

(2) The basic building blocks, divided into two parts:

  (a) Nouns, representing data to be retrieved, entered, or operated on.

  (b) Verbs, representing operations to be performed on the data.

(3) An interpretive compiler, allowing the user to put together instructions or queries.

The feasibility of the building block approach and of keyboard programming by the user was demonstrated, and the applicability of available display/control consoles suitable for providing convenient man-machine communication was verified.

One key product of the research to date has been the development of a basic methodology for implementing on-line systems. The conflicting goals of machine independence, on one hand, and problem independence, on the other, were accepted as desirable characteristics of an ideal man-machine interface. A machine-independent interface would be adaptable, with a minimum of costly and time-consuming revisions, for use with whatever hardware might be available to the user. A problem-independent interface would provide the user with a sufficient degree of flexibility and power to contend with a continually changing set of needs without obliging the system developers to anticipate these needs in every detail. Both of these properties would make a single interface suitable for a broad class of user communities and problem areas without substantial modification.

A solution to these problems is suggested which is adequate for the current research and development requirements and which should provide a firm basis for future methodological advances. This approach involves a division of the interface (and its chronological development) into three distinct packages:

(1) Base package, consisting of hand-coded, machine-language subroutines of the utmost simplicity and generality. This package provides, in effect, a low-level assembly language for use by persons with considerable programming experience. This package is entirely problem-independent.

(2) Procedure-oriented package, consisting of a number of subpackages of subroutines of increasing specialization, composed entirely of base-package components. This package provides a set of increasingly specialized and sophisticated design languages for use by systems design personnel with considerable programming knowledge. The procedure-oriented package is entirely machine-independent.

(3) Problem-oriented package, providing an open-ended, strongly problem-oriented set of subroutines out of which members of the intended user community will construct their own operationally-adequate problem-solving system.

This report which is the first part of a two part report covers the first 18 months of the contract or roughly through November 1964. Part Two which will be published in December 1965 will cover the period November 1964 and November 1965.

II. Introduction

The parameters of the modern military environment are such that man's reaction time and capabilities are no longer adequate to cope with the situation alone. The response time available to react to the threat of a modern high performance aircraft or missile is short enough to require automation of many control functions previously handled by manual means. The danger of escalation of an incident into a major conflict, and the resulting requirement for more detailed information at higher levels of command increase the amount of information to be handled by all echelons of command. This great increase in the amount of information to be handled, coupled with the requirement for rapid response, imposes the need for automated assistance in the information handling function.

The initial efforts in automating command and control systems were in the area of weapon control and tactical operational systems. The NTDS system is a good example. In recent years considerable attention and funds have been expended on what might be called strategic operational systems. Strategic is used here to describe those command and control systems which are concerned primarily with planning for future operations. Specified Commands such as CinCLANT and CinCPAC fall into this category. This investigation concentrates on the strategic or Command Information Systems. In most Command Information Systems there are two major problem areas: information acquisition and information utilization. The major emphasis of this work has been toward providing methodology for the use of computers to assist the military in information utilization.

Most current systems have enormously large and rigid programs as part of their data processing function. The user interacts with these systems in a completely pre-determined way. That is, the system designer has thought up as many as possible of the answers which the commander *may* require from the system and then has programmed backward to define the commander's (or his staff officer's) input (query) which would result in retrieval and/or manipulation of the appropriate data. The more advanced systems have a defined "query language" to initiate the inquiry process which is more like English than a coded computer instruction. These are still *quite inflexible* in character; must be completely pre-programmed; and do not allow the user to *control* the level of detail of the answer, the selection of alternative logical processes based on preliminary results, or the relevance of the material presented. Making changes to existing large system programs in response to a changed(and usually unexpected) military situation, to the introduction of new weapons, or to changing command personalities is a difficult and time-consuming task because of the user-systems analyst-programmer-machine communication chain. It is in these very areas of *inflexibility of programming* and *difficulty in communicating* where the "on-line" techniques show the greatest promise for improvement.

III. On-Line Systems

A. General

In the most general sense, an on-line data processing system (as we shall use the term) comprises:

a- a central processing computer

b- a man-machine communications interface

c- peripheral I/O and storage facilities

d- an operator expert in the field from which the data to be processed is derived.

These terms will require some elaboration to render explicit what we have in mind in talking of on-line data processing systems. The central processing computer might be any of the various general purpose digital main frames with which we are familiar. It need only be compatible with the man-machine communications interface sub-system to be employed. Included with the central processing computer we assume there is a working memory facility, internal or external to the main frame, of reasonable size and speed.

The heart of the on-line system is the man-machine communications interface. This may be viewed as consisting of two conceptually independent parts, a collection of command and data keys by which the operator controls the computational processes, and an output device (or set of such devices) by which the hardware communicates to the operator.

B. On-Line Objectives

There are in existence today, many different systems which are called "on-line" systems. Each system designer had as his goal a specific application and hence the term "On-Line" means something slightly different to each. In general, however, there is one specific objective of most of these systems. That objective, simply stated, is to put the user (man) directly in the system loop so that his experience and intuition can be conveniently coupled with the powerful and rapid computational capability of the computer in such a way that there is an optimal balance between capabilities.

The key words in the statement are "directly" and "balance". In previous implementations the user was faced with a choice in his attempt to communicate his problem and his methods for solution to the computer. He could either become a programming and computer expert so that he, himself, could communicate; or, more often, since most users have neither the time nor the desire to become programmers, he was forced to communicate through an interpreter or programmer. Most programmers are experts in computer language; however, they are not experts in the user's particular field, hence, the experience of the user often gets lost in the translation. The user wishes to communicate "directly" with the system himself.

The technology upon which most Command Information Systems have been designed and implemented grew out of experience with control systems such as SAGE. There has been a natural tendency to use many of the same ground rules which have resulted in using the computers to perform functions which are difficult and costly for a computer to perform, but which can readily be performed by man. In work performed recently by Dr. Gagliardi, of Dunlap Associates, (1) he found that while it was

(1) "Development of Man-Computer Systems for Solving Targeting Problems". Presented at Western States Navy Research and Development Clinic, Boseman, Montana, July 1964.

possible to program a 7090 to perform some particular functions in total, it was cheaper and faster to allow a man to perform most of the functions and use the computer to do only the tedious, routine calculations. The user must have the ability to control the "balance" of functions performed by himself and the computer if the on-line system is to provide the flexibility to serve as a tool for different users with different problems and different methods of attack.

Perhaps the best known of the on-line systems today is the MIT CTSS System of Project MAC. Other systems include the SDC Time-Sharing System, IBM's QuickTRAN, the JOSS System at RAND Corporation, the Stanford Time-Sharing System, the Bolt, Baranek and Neuman Hospital Computer System, and the STL Scientific On-Line Center. In addition, many of the Universities such as the Moore School of Engineering of the University of Pennsylvania, the University of California at Santa Barbara, Dartmouth University and UCLA either have systems or are planning systems.

None of the above systems meet all of the desired characteristics that an on-line system should provide. The major reason is that, for the most part, the initial effort has been directed toward time-sharing of a large system by many users. Secondly, most of the systems have been designed with a highly technical and programming oriented user in mind, rather than a non-programmer. Hence, the language of communication between the user and the system has usually been a programming language such as: FORTRAN, JOVIAL, LISP, ALGOL or modifications of these languages.

Considerable effort must still be expended to facilitate flexible man-machine communications between a non-computer oriented professional and a data processing complex in non-technical environments.

C. Bunker-Ramo On-Line Techniques

The Bunker-Ramo Corporation has been working in the area of On-Line techniques for several years. The initial work was sponsored by Rome Air Development Center and began in 1962. This effort resulted in the Culler-Fried On-Line Scientific System. There quickly emerged during this effort a highly flexible man-machine processing system whose organization shows many of the points we have raised in the foregoing discussion.

Briefly, this system employs two keyboards through which the operator controls all computational processes; the results of these processes are displayed to the operator on a CRT in alpha-numeric or graphic form. One of the keyboards consists of process control keys, or operators, and the other consists of data keys, or operands. There are several levels of operator keys and data keys. Through this separation of keys into levels, provision is made for the incorporation into the system of a great number of individual keys.

The operator keys are labelled with familiar mathematical symbols denoting a variety of fundamental operations; the data keys are labeled with letters of the alphabet, suggestive of mathematical variable designators. In general, the evaluation of a mathematical expression is accomplished by the successive depression of operator and operand keys in the order in which they appear in the expression. Intermediate or final results can then be displayed on the CRT for the user's evaluation and interpretation. Mathematically meaningful expressions can be directly converted into permissible sequences of operator and operand key depressions at the system console; thus, anyone with adequate mathematical background can make use of the Culler-Fried system with a minimum of special training and the possibility of user "errors" is minimized; most important of all, the system's user is able to concentrate his attention on the mathematical problem at hand rather than on an intricate processing language structure.

Flexibility in the Culler-Fried mathematical system is provided through the system's console programming feature; this feature permits the user to construct a virtually unlimited number of specialized operator keys, and to modify these at will. There is no limit to the amount of cross-referencing and embedding that is allowed in the construction of console program keys, and as a result, single operator keys of tremendous power (whose ultimate constituents are always members of the basic set of console programs) will generally be built up during a single user's experience with the system. Once constructed, the individual user's complete set of console programs can be quickly read out onto magnetic tape for permanent storage. At the beginning of each session with the system, this user is then able, with a minimum of delay, to operate with the full range of programs he has specially tailored to his purposes.

It should, of course, be pointed out that the problems facing this system's designers were considerably less formidable than those which will be encountered outside the domain of mathematical applications. In the first place, the rigid formality of mathematics itself greatly reduced the problems of analyzing the requirements of the intended user community. It could be assumed that this user community was familiar with, and would be bound by, the conventions of modern mathematics. In the second place, and of equal if not greater importance, the system designers did not have to be concerned with a large data base or with the inherent file maintenance and search and retrieval problems.

This system was operated experimentally for two years. The experience gained from the Culler-Fried system led to a set of characteristics which is believed to be basic. These are:

1) There should be direct and convenient two-way communications between the user and the computer, with a minimum of intermediate people, procedures and programs. The user should be able to communicate without the necessity of learning programming languages.

2) The computer should respond "immediately" insofar as the user is concerned.

3) There should be the capability of continuous interaction or interplay during the problem-solving process.

4) The user should have the capability of structuring the problem-solving process and changing its structure at will. This requires that he be able to "program" with the tools provided him - - a keyboard using his own vocabulary in "English-like" statements.

In designing an on-line system the importance of the particular user must be emphasized. The realization of the potential advantages of on-line computation depends on the user's expertise in the subject matter field from which the problem to be solved is derived. For he will himself develop the higher levels of the processing language represented through the interface sub-system to suit his own needs in extracting the information he requires from the raw data. Ultimately, his own recognition of reasonable procedures in the processing of this specialized data will become embedded in the syntax of the processing language itself. And since this syntax is a function of his own competence in the subject matter field, the development of an efficient processing language will in the end depend upon the degree of that special competence he possesses.

The principal problem faced by the designer of an on-line data processing system is the specification of the initial configuration of the interface sub-system. This problem involves three interrelated tasks:

a- specification of foundation sub-routines to be associated with the basic set of interface command keys

b- specification of the addressing function of the basic data keys

c- provision for the system's growth.

The specification of foundation sub-routines is the most crucial of these tasks. First of all, a study must be made of the data processing requirements of the user community for which the on-line system is intended. The results of this study should be a small set of very general, relatively low-level foundation sub-routines which satisfy the data processing requirements of the intended user community and which are, to the greatest possible extent, mutually exclusive in their functions. The very nature of this task suggests that there is no formal procedure for its accomplishment. Sound judgement, deep understanding of the subject matter field, and a faculty for insight into organizational problems are all called for, and the adequacy of the results can only be confirmed through empirical testing.

The syntax of this initial configuration must be, to the greatest possible extent, natural; i.e., the rules governing legitimate usage of the command and data keys must correspond, as much as possible, to the operator's notions of what sort of things can and cannot be done with the data at his disposal. To put it another way, forbidden sequences of basic operations should correspond to meaningless or improper statements within the subject matter field familiar to the user.

IV. Application of On-Line to Command and Control

A. Command Information Systems

To begin this discussion of the problems to be dealt with in automating Command and Control functions, we should examine some of the outstanding characteristics of the modern Command and Control community. We shall concentrate in this section on those problems which are of particular relevance to command staffs and shall not concern ourselves with problems peculiar either to the other services or to strictly weapons-control aspects of the Navy's overall Command and Control structure.

Because of the high speed of modern weapons delivery systems, the long (and increasing) range of reconnaissance and surveillance systems, and the growing interdependence among geographically distant areas in the realm of mutual defense, a primary requirement of an effective command staff is the ability to handle very large volumes of information at high speeds. The need exists for ways to collect, catalogue, store, and selectively retrieve this information on demand. Another requirement that must be met is that of performing a number of (generally elementary) computations on portions of this mass of raw data; here we are referring to direct mathematical operations such as statistical analysis or navigational computations.

These two major information-handling functions are subordinate to the principal goals of the command staff: to supervise the planning, training, support, deployment, and operation of forces within its area of responsibility. All of these sub-functions involve extensive and complex decision-making procedures.

During the initial portion of the contract period, we visited several Naval Commands to obtain information. CinCLANT was utilized as the prime source of information. Our objective for these visits was to determine problems faced by the military today in automating their command information function, and the limitations represented by the current state-of-the-art in information handling. We had, rather naively, expected to obtain enough information to enable us to pin-point those problem areas where on-line techniques could be used to overcome some of the limitations of conventional information processing.

We found that such information was not readily available. The military personnel we talked to were most helpful. The problem is that while the average military operational person is extremely competent in his field, he simply does not know enough about the data processing field to understand what assistance automated information processing can provide him in the performance of his daily duties. Hence, he is unable to define his requirements for an information processing system.

Three characteristics of present-day information processing technology present obstacles to its adaptation to modern command staff information handling applications: the inflexibility of the man-machine communication, the need for special operating skills, and dissimilarity of environments.

The normal man-machine communication medium in most automated information systems consists of a variety of more or less rigidly structured programming languages with varying design bases. However effective these languages may be in facilitating programmer-machine communication for special purposes, their utility in the command staff environment is greatly restricted by their requirement for training on the part of the user, and by their built-in orientation toward narrowly specialized problem areas. FORTRAN, while quite simple to learn and use, is not very well suited to symbol-manipulation procedures and plain text processing, while ALGOL, which offers more general capabilities, is more difficult both to implement and use. None of the familiar programming languages possesses to any high degree the quality of "naturalness" (with respect to the C + C community) by which man-machine communication is greatly enhanced.

For most information handling systems a broad array of operating skills is required, and this requirement itself poses an impediment to optimal man-machine communication. One of the ways in which this hindrance is traditionally overcome is through specialization of personnel functions- -- operational personnel encounter problems, systems analysts reduce these to computational terms, programmers flowchart this intermediate result for a particular computer system, coders produce machine instruction lists from the flowcharts, keypunch operators prepare machine-readable copies of the instruction lists, and, finally, machine operators "run the problem". At the end of the problem run, this sequence of specialized intermediaries is again encountered in reverse (with some omissions, such as the keypunch operators) ---the programmer helps interpret the results for the systems expert, who then explains what has happened for the operational specialist, who, at last, interpretsthe results at the level of actual application. Many skills are required to maintain the integrity of this complex loop and, precisely because the range of skills is generally greater than individuals care (or are able) to master, it is difficult to significantly reduce the number of links that make up the "long-distance" man-machine communication chain. As a result, communication is error-prone and slow.

The environment for which most contemporary information systems were designed is often not comparable to that in which the command staff must operate. In particular the constraints of response time, data volume, and "data half-life" are more severe in the military command staff environment than in many other areas of application. Consider the relative leisure with which a physicist, for example, may pursue the solution to a given theoretical problem; he has the advantage of being permitted numerous trial attacks on his problem and of developing a body of data sufficient for the problem as it is required. Moreover, his main emphasis is likely to be on method rather than individual results (which, in fact, is the reason why so leisurely an approach is justified). These working conditions contrast sharply with those that confront the command staff which must make "best available" decisions on the basis of incomplete information (which itself must be quickly extracted from great quantities of irrelevant materials) under urgent time constraints, and in a problem are so raggedly structured as to render most problems more dissimilar than alike.

A schematic illustration of some of these considerations appears in Figures 1 and 2. Figure 1 portrays the fragile and slow chain of communication that separates operational personnel from the data processing system in typical applications. It should be borne in mind that modifications in this chain, when required by changing problem patterns and trends, can be costly indeed in terms of dollars and time; it may take months of intensive efforts to "re-tool" such a system to handle new problem types or to achieve greater operating speed.

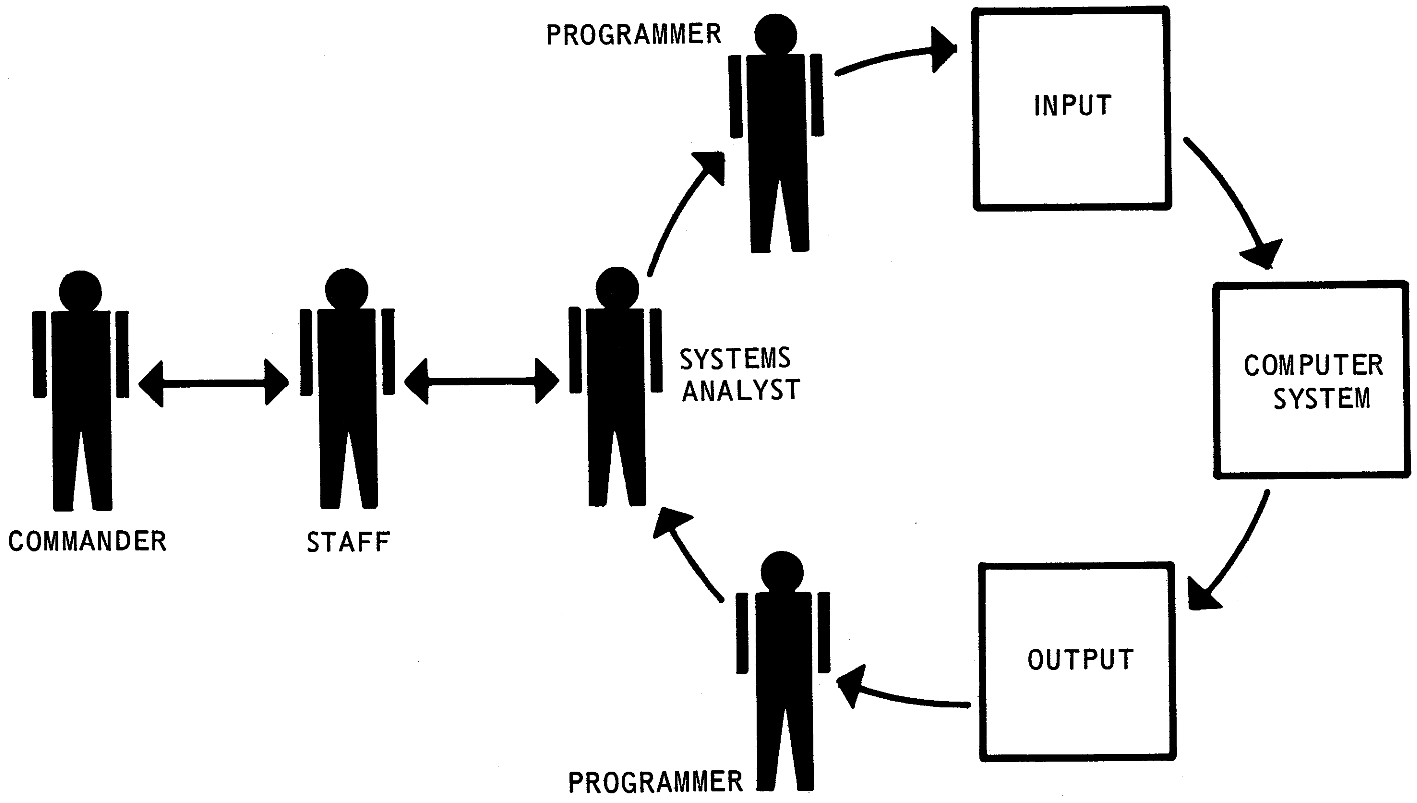


FIGURE 1 CONVENTIONAL SYSTEM STRUCTURE

In Figure 2 the simplification in the communications chain through on-line techniques is illustrated. The Bunker-Ramo Corporation's efforts to design and test systems utilizing these techniques, based upon the principles discussed earlier in this report, are described in the following sections.

B. <u>On-Line Experimental System</u>

In the course of the early portion of the study, we sought to produce a workable definition of the term, "On-Line" as applied to man-machine interaction, to justify the characteristic features of such on-line systems; to determine classes of command problems to which the on-line technique is well-suited; and to develop skeletal design parameters for a relatively small scale on-line experimental information processing system. The knowledge that was accumulated as a result of these early studies, led us to record the feasibility of application of on-line techniques to certain problems in command and control as established.

To test the validity of this conclusion, a small scale on-line system was developed.

Before discussing the features of this system, it is necessary to describe the equipment complex so that the reader can better visualize the discussions of man-machine interplay.

<u>Bunker-Ramo On-Line Experimental Center</u>

The equipment complex used consists of the following:

Computer	AN/UYK-3
Display/Control Console	BR-85
I/O Controller	BR-141
Magnetic Tape Controller	BR-192
Magnetic Tape Unit (2)	BR-170

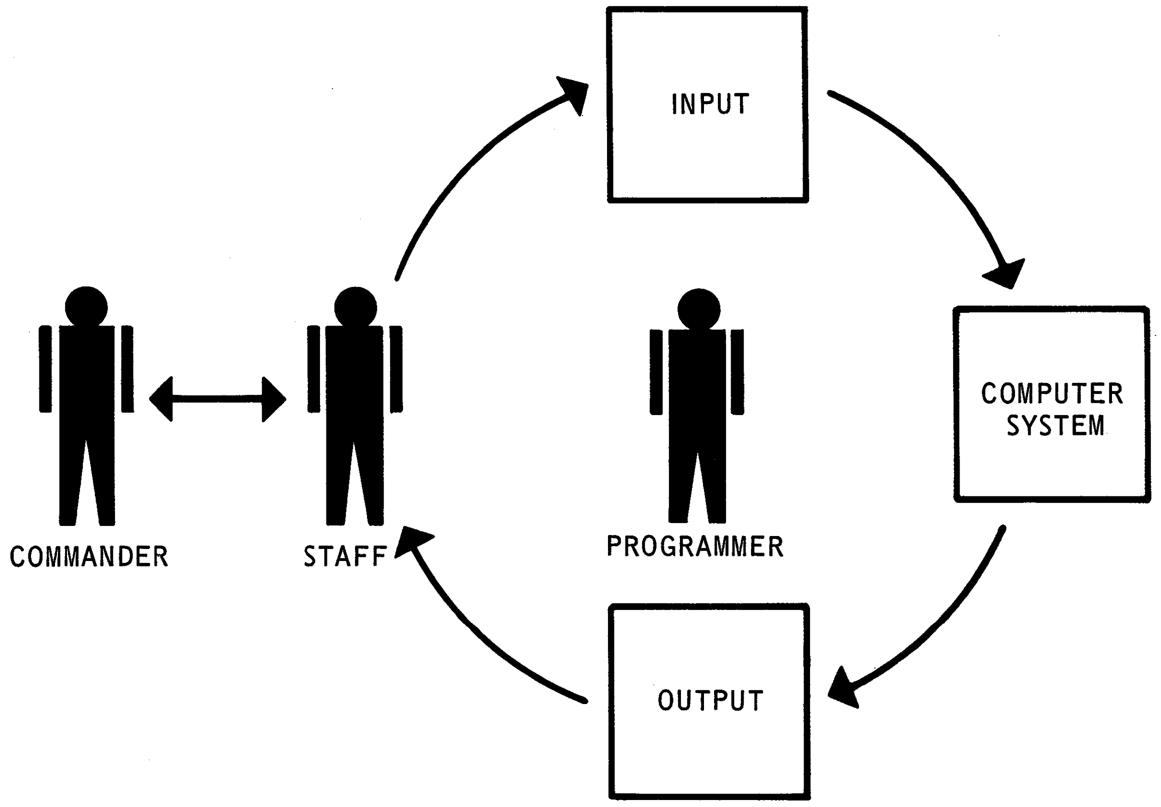


FIGURE 2 ON-LINE SYSTEM STRUCTURE

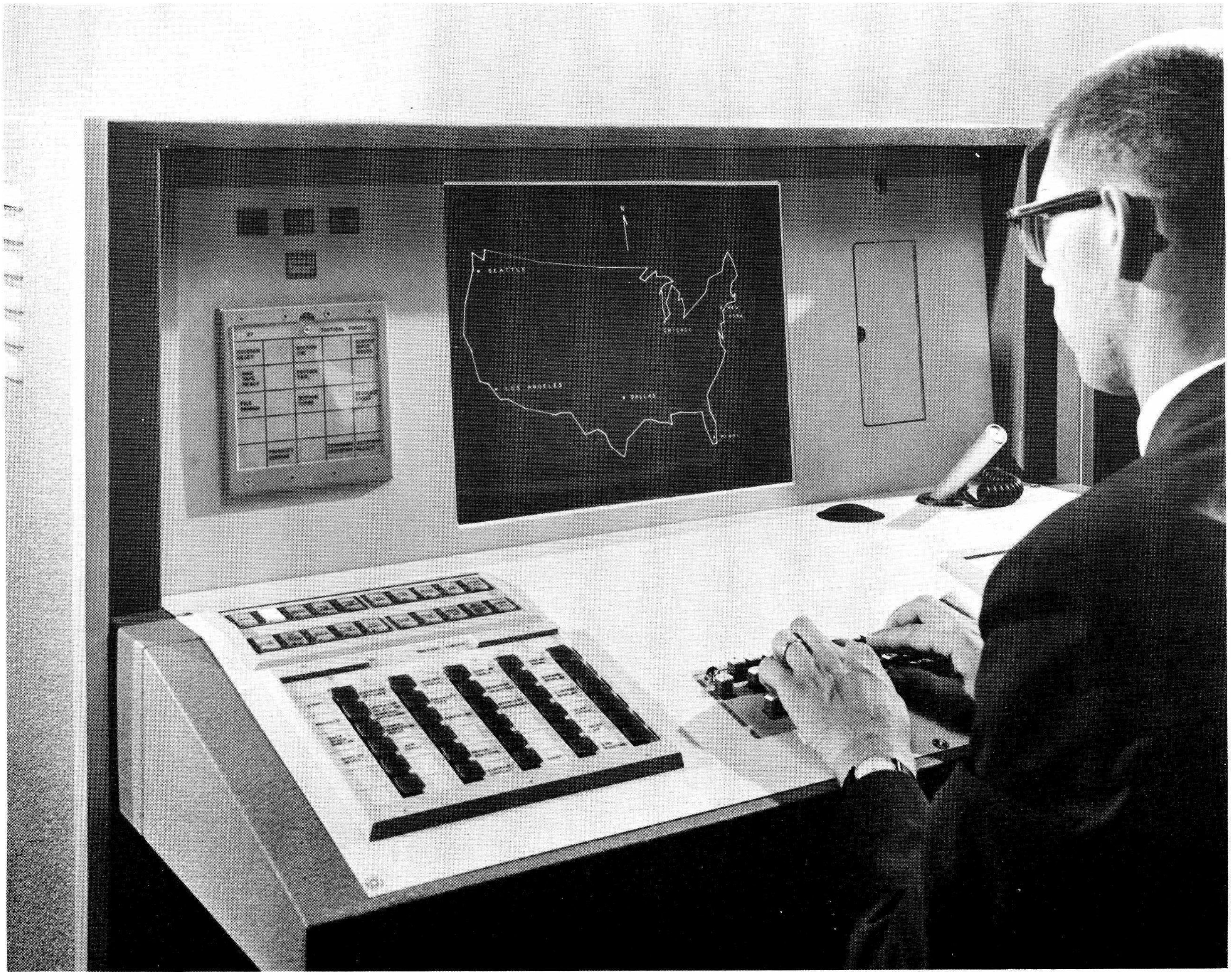
The computer is a 16K, 15 bit stored logic computer with a 2 micro-seconds memory cycle. The BR-85 Display/Control Console is shown in Figure 3.

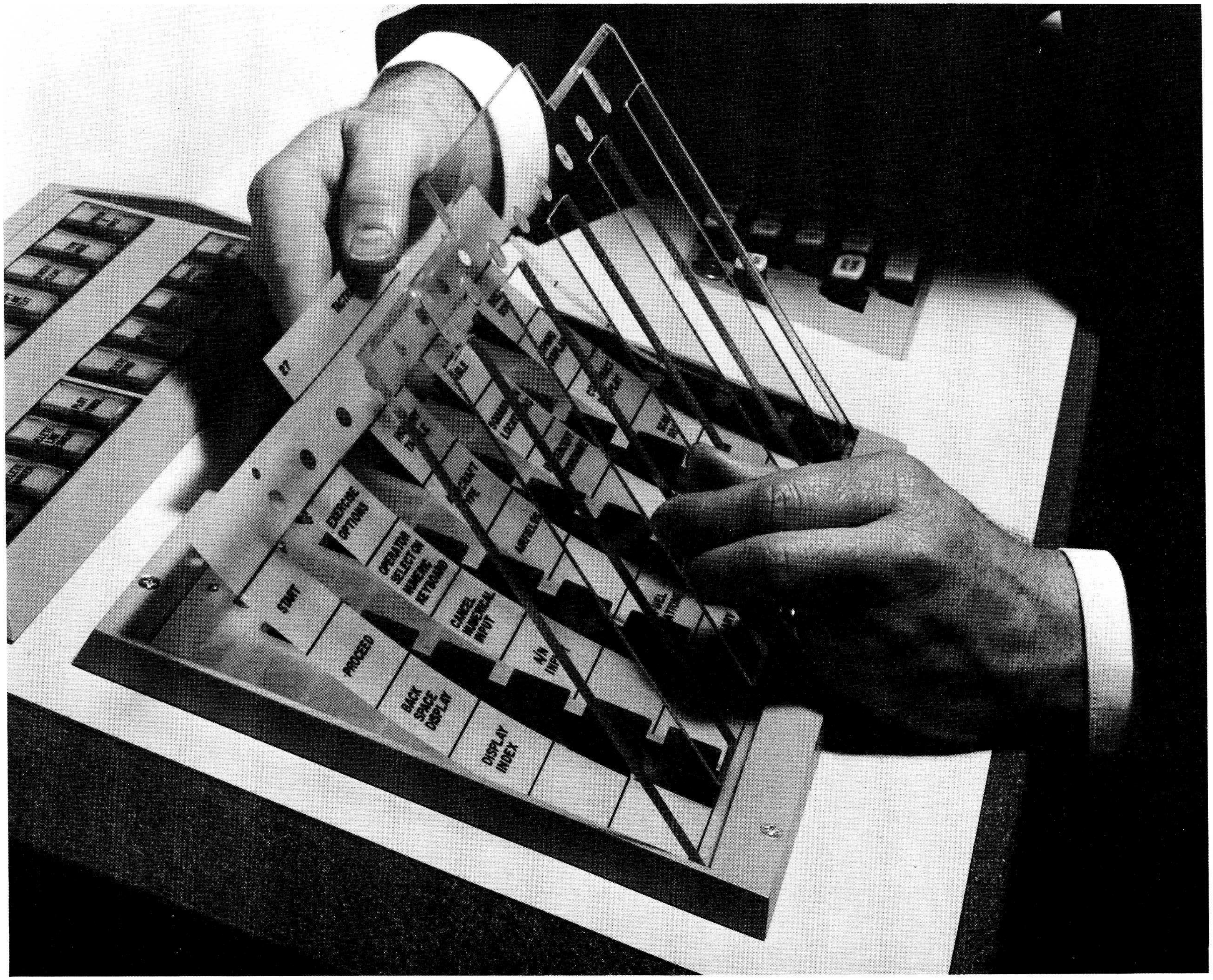
Text, symbols, point plots and lined drawings may be produced in the 12-inch by 16-inch active display area on a 23-inch aluminized TV-type screen. Each display element may be placed at any one of 512 horizontal or vertical positions. Up to 32 lines of 64 alpha-numeric symbols may be placed on the screen. Symbols are generated at the rate of 100,000 per second, may be produced in two different sizes, and may be caused to blink on the screen.

A 4096 word, 9 bits per word magnetic core memory retains the information for the display and refreshes it at either 30 or 60 cycles per second. The computer has random access to the memory and can transfer blocks of words into or out of it at 100,000 words per second.

30 keys, labeled by thin, interchangeable, plastic overlays, are used to send messages to the computer. The interchangeable overlays, (see Figure 4) permit any one of 64 (or 128 with optional second program keyboard) different pre-programmed routines to be activated. A light next to each program key is identified by the overlay and independently controlled by the computer. These lights may be programmed to cue the operator as he uses the keyboard.

A keyboard is provided to enter alpha-numeric information into the memory, from which the data is displayed on the screen. The shift, carriage return, backspace, and advance keys operate like those on an electric typewriter.


N
SEATTLE
CHICAGO
NEW YORK
LOS ANGELES
DALLAS
MIAMI
TACTICAL FORCE


27
EXERCISE OPTIONS
START
PROCEED
OPERATOR SELECT ON NUMERIC KEYBOARD
CANCEL NUMERICAL INPUT
BACK SPACE DISPLAY
DISPLAY INDEX
A/N INPUT
AIRFIELD

20 keys and lights are provided to assist the operator in off-line message composition and editing. The operator may create, delete, or change the position of line segments or point plots; he may initiate a typewriter mode of operation and copy or delete whole words or lines of text; he may display different portions of the memory or clear portions of the memory. The keyboard also has control and information functions related to the light gun, the cursor, and the status of the console.

An electronically generated crosshair may be produced on the screen and may be moved to any position through use of the cursor control. The coordinates of the cursor may be used internally or sent to the computer.

The operator may select any display element on the screen with the light gun. Operator aiming is verified by blinking of the element detected by the light gun. The address in memory of the word producing that display element may be used internally or sent to the computer.

25 status lights furnish the operator with status information and other fixed messages.

Initial System Implementation

The initial or prototype system, while limited in scope, illustrate many of the design features we have discussed. This system was demonstrated in the lobby of the Main Navy Building in Washington, D.C., in June of 1964.

It was of importance in the development of this system to analyze the typical functions of the naval command staff in order to (1) isolate those functions most immediately amenable to automation, to receive our attention in this first-level effort, and (2) reduce these chosen functions to a number of basic, clearly-statable "building blocks" capable of being programmed for the hardware at our disposal. Here, the modesty of our goals in this first effort enabled us to circumvent many of the more formidable problems which will be faced eventually as a deployable system is developed. We concentrated our attention primarily upon elementary data-retrieval problems and the simplest of useful mathematical operations. A number of data categories appropriate to naval problems were listed, pertaining both to ships and to ports. To aid user acclimation to the system, these data categories were assigned as the labels of our data keys.

The selection of operator functions was again made with a view to attaining the greatest degree of simplicity in the system's structure compatible with our principal goal of demonstrating considerable flexibility in retrieval of desired materials from the data base. The decision was made to permit the user to specify the search parameters through the console's input alphanumeric keyboard and to combine groups of such criteria by employing a set of logical operator keys. Thus, the system's data keys were made to correspond, one-to-one, to the data file categories; the system's operator keys were made to correspond to elementary logical connectives (such as "and", "or", "greater than", "less than", "equals", etc.) and to a small set of direct output specifications (such as "display tabular", "display graphic", etc). These made up the basic "building blocks" of our prototype on-line system.

Simple minded as these basic functions were, quite complex single operator keys could be constructed through console programming. In this first-level system three operator keys were reserved for console programming;

the basic features of the system's console programming can be described in terms similar to those used above in the discussion of the Culler-Fried system. While considerable power could be built up, through this means, under a single console operator key, a good deal of ingenuity on the part of the user was demanded in many instances to achieve his particular ends. This was due to the somewhat inelegant syntactic structure of the processing language embodied in the system; a number of rather artificial conventions had to be borne in mind by the user, and certain more or less covert functional conflicts could be generated through the utilization of innocent-appearing operator key sequences. These difficulties can be summed up simply in stating that our console processing language was not sufficiently "natural". The importance of this elusive property of "naturalness" in on-line data processing languages was forcefully brought home to us through our experiences with the prototype command staff system.

The data base for the demonstration consisted of information on 15 ships--10 of which represented a U.S. Naval Task Group--and 10 ports, all in an imaginary area resembling the Mediterranean. Figure 5 shows the items of information which were available on each ship and port.

Our method of operation for the demonstration was to divide the basic building blocks into action items and operations which could be structured into questions or instructions by depressing keys. The action items or nouns shown in Figure 6 for the most part represent the file categories or specific items of information. Notice that we have some buttons or keys which are labeled assignable nouns. These are for temporary storage. Also notice that some of the buttons have dual meanings which are distinguished by prior depression of either the <u>ports mode</u> or <u>ships mode</u> key at the lower right.

DATA BASE

SHIPS FILE	PORTS FILE
NAME:	NAME:
TYPE:	REPAIR:
LOCATION, E:	LOCATION, E:
LOCATION, N:	LOCATION, W:
DISPLACEMENT:	COUNTRY:
ASSIGNMENT:	SECURITY:
SPECIAL CAPABILITIES:	ACCESSIBILITY:
SUPPLIES:	REPLENISHMENT:
PORTS SCHEDULED:	S & R:
CASREP:	SPECIAL PROBLEMS:
MAX. SPEED:	FLAGS IN PORT:
MISSION PRIORITY:	
READINESS:	
CAPTAIN:	
PASSENGERS (EMERGENCY):	
COURSE:	
SPEED:	
ID:	

FIGURE 5 DATA BASE

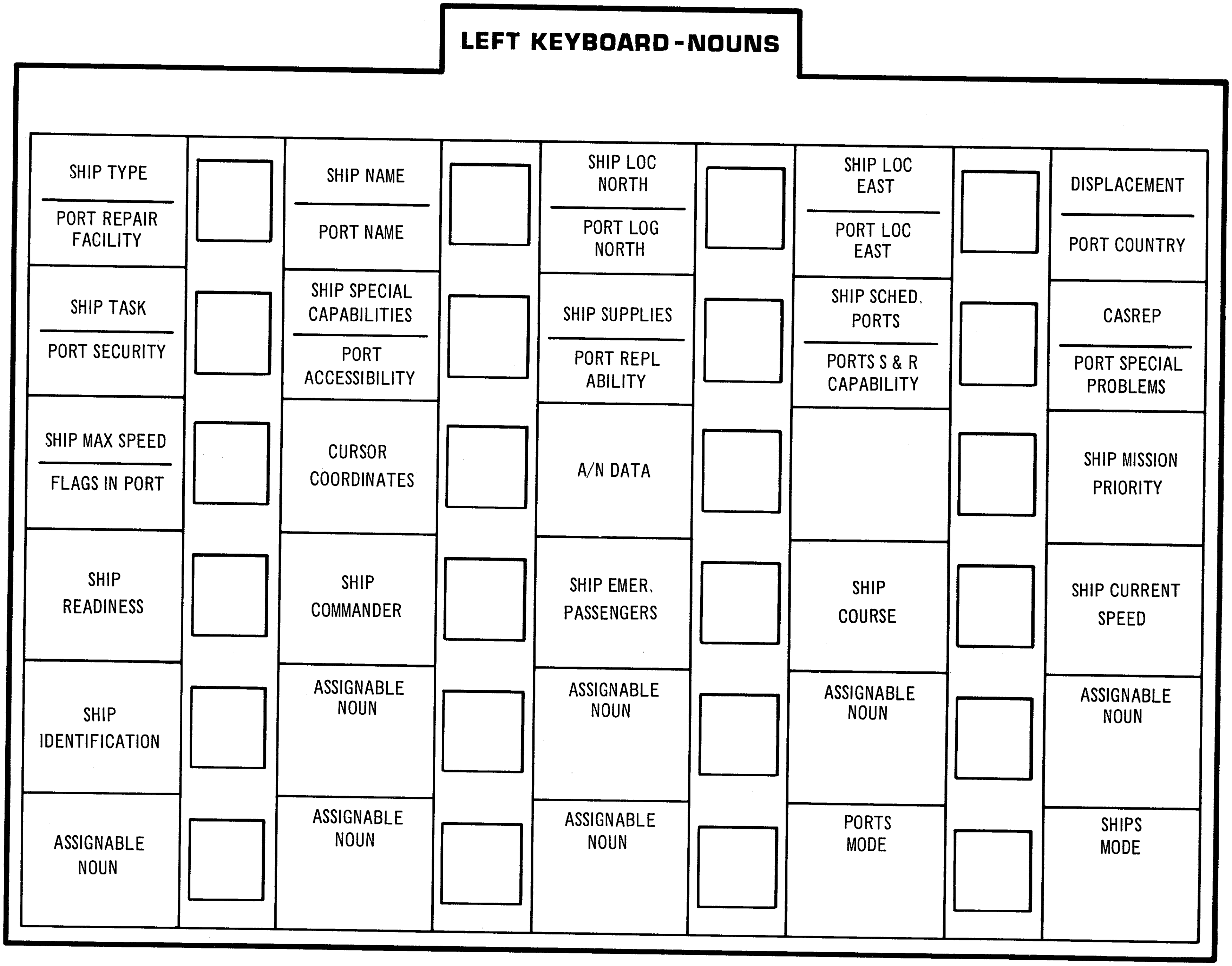


FIGURE 6 LEFT KEYBOARD – NOUNS

The operations or verbs shown in Figure 7 show the basic operations which are available. Notice that there is a hard copy button which allows printout of the requested information as well as a CRT display. There are two special buttons which are of a higher level. These are "Distance Between" which computes the distance between two locations and "Location Within" which finds all ships or ports located within a defined area by the user. The Program A, B, and C buttons located in the upper right are programmable from the keyboard by combining a group of button pushes.

The low level of the basic operations was chosen purposely to show that fairly complex operational functions can be built up from these few basic operations. As an example of how the basic elements can be combined in the user language, a typical operational function which we used in the demonstration was "Find the closest port to a given ship". This was programmed from the keyboard and stored under one of the program buttons because of the frequency of use of this function. The total number of steps was 27. Once this program was stored we could then select a ship and find the closest port to that ship by depressing only the one button, or we could add criteria such as finding the closest port having a repair capability large enough to handle a particular ship.

This example points up the reference made to the difficulty of determining the proper level for defining the basic elements on building blocks. In the example shown 27 steps or button pushes were taken to structure the desired results. If we had not had the special button "distance between" the number would have increased. On the other hand, if we had added a few more of these higher level special buttons such as "find the minimum" the number of steps would be significantly reduced.

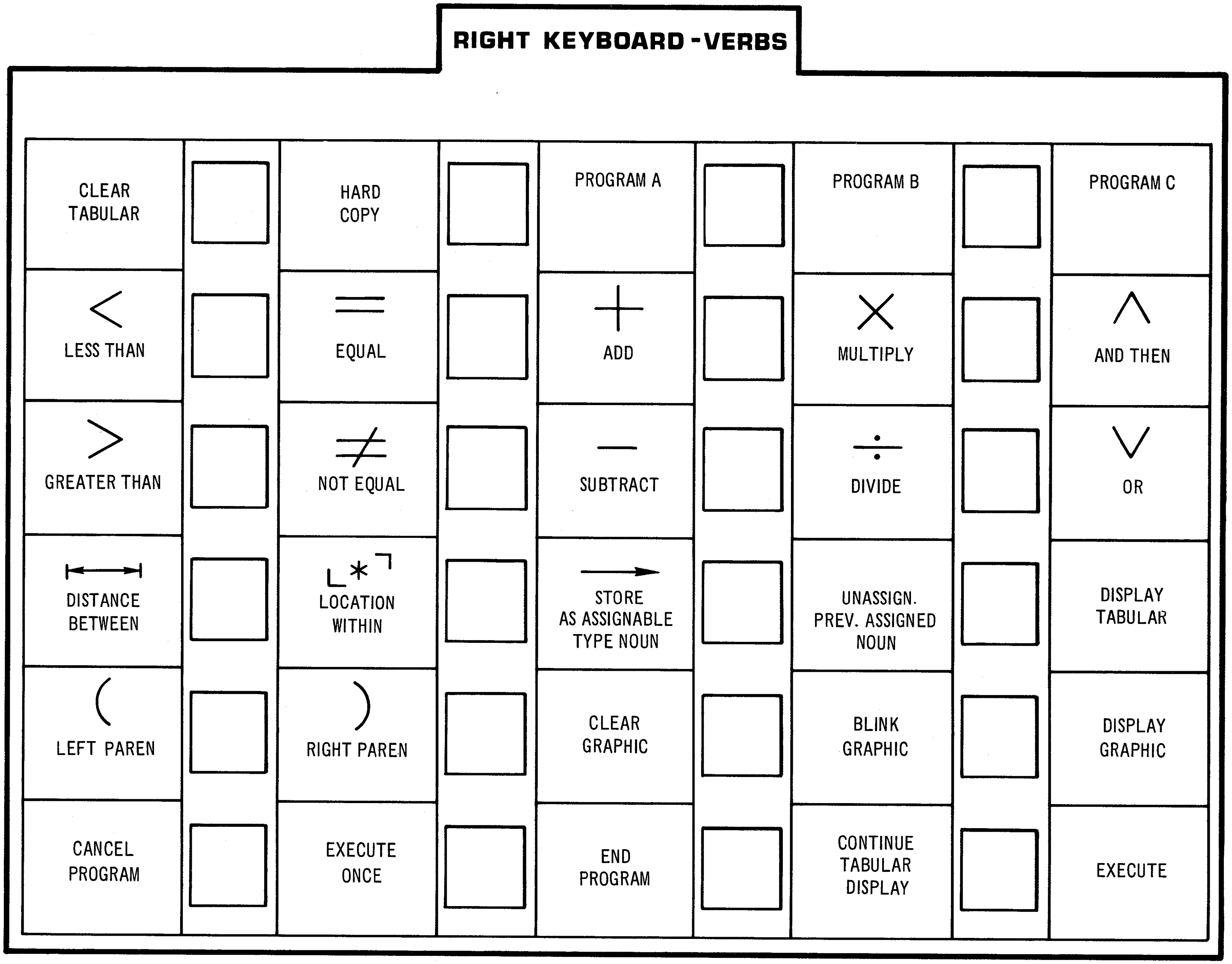


FIGURE 7 RIGHT KEYBOARD – VERBS

A significant aspect which was demonstrated was that of more optimum interaction between the user and the information processing system. During the demonstration we had the opportunity to have several military personnel solve a hypothetical problem using the system and the data base. It was interesting to note that each one attacked the solution of the problem in a slightly different way and used the system in different ways. Some did quite a few calculations in their heads. Others used the systems to do these calculations. The system was flexible enough to allow each user to solve the problem in his own way.

The initial system consists basically of an interpreter of 1200 words, and a compiler of 1500 words. Flow charts for this system have not been included as part of the report but are available.

Improved System Implementation

The initial implementation demonstrated the feasibility of a military operational user interface. It also pointed out the need for a more flexible system.

There are many different levels of users in a command information system, just as there are many different types of problems to be solved. For example, a user might be a programmer, a systems analyst, duty operations personnel, or staff personnel concerned with planning. The requirements imposed by each of these on command information systems will differ, and more important, different terminology will be needed by each to communicate with the computer. A programmer will want to converse in some type of programming language. The operations or planning personnel will wish to converse in command and control terminology. A true on-line system should be flexible enough to serve all of these users. Therefore, there is no one set of basic building blocks that will practically serve all users.

One key product of the research has been the development of a basic methodology for implementing on-line systems. The conflicting goals of machine independence, on one hand, and problem independence, on the other, were accepted as desirable characteristics of an ideal man-machine interface. A machine-independent interface would be adaptable, with a minimum of costly and time-consuming revisions, for use with whatever hardware might be available to the user. A problem-independent interface would provide the user with a sufficient degree of flexibility and power to contend with a continually changing set of needs without obliging the system developers to anticipate these needs in every detail. Both of these properties would make a single interface suitable for a broad class of user communities and problem areas without substantial modification.

A solution to these problems is suggested which is adequate for the current research and development requirements and which should provide a firm basis for future methodological advances. This approach involves a division of the interface (and its chronological development) into three distinct packages:

(1) Base package, consisting of hand-coded, machine-language subroutines of the utmost simplicity and generality. This package provides, in effect, a low-level assembly language for use by persons with considerable programming experience. This package is entirely problem-independent.

(2) Procedure-oriented package, consisting of a number of sub-packages of subroutines of increasing specialization, composed entirely of base-package components. This package provides a set of increasingly specialized and sophisticated design languages for use by systems design personnel with considerable programming knowledge. The procedure-oriented package is entirely machine-independent.

(3) Problem-oriented package, providing an open-ended, strongly problem-oriented set of subroutines out of which members of the intended user community will construct their own operationally-adequate problem-solving system.

Several advantages of this methodology are at once obvious. To move a given full system, for example, from one central processor to another, one must only recode the base package; the rest of the system can remain substantially intact. On the other hand, if the requirement is to prepare for a new problem area and/or a new community of users while retaining the same central processor, only the problem-oriented package and a greater or lesser portion of the procedure-oriented package need be reprogrammed; the problem-independent base package is used in its entirety. Furthermore, this reprogramming involves no machine-language coding but can be accomplished through the highly efficient techniques of console programming.

The base package has been completed and is described in detail in Appendix A. A description of the improved user level will be included in Part Two of this report.

BIBLOGRAPHY

"Application and Implementation of Deacon-Type Systems" - General Electric Company, Santa Barbara, California

"Deacon Breadboard Processing" - General Electric Company, Santa Barbara, California

"Lap-List Assembly Programming System"- General Electric Company, Santa Barbara, California

"Deacon Breadboard Grammer"- General Electric Company, Santa Barbara, California

"Deacon Breadboard Summary" - General Electric Company, Santa Barbara, California

"JOSS: A Designer's View of an Experimental On-Line Computing System" - J. C. Shaw, Rand Corporation

"Research on Heuristic Problem Solving Machines" - Westinghouse Electric Corporation

"Interim Technical Report:" A Statistical Optimization of Search Time in an Information Retrieval System - Philip Leslie Leifer, University of Pennsylvania, Philadelphia, Pa.

"Transition to Command and Control Systems" - Parker L. Folson, University of Calofirnia

"Studies for the Design of an English Command and Control Language System" - Arthur D. Little, Inc. Cambridge, Mass.

"Interim Technical Report:" On a Study of Information Storage and Retrieval" - D. S. Sharp and J. E. McNulty, University of Pennsylvania, Philadelphia, Pa.

"Verbal and Graphical Language for the AED System; A Progress Report" - Project MAC, Massachusetts Institute of Technology, Cambridge, Mass. - Doublas T. Ross and Clarence G. Feldman - May 1964

"Stress: A Problem-Oriented Language for Structural Engineering" - Project MAC, Massachusetts Institute of Technology, Cambridge, Mass - John M. Biggs and Robert D. Logcher - May 1964

"OPL-I An Open Ended Programming System Within CTSS" - Project MAC - Massachusetts Institute of Technology, Cambridge, Mass. - J. Weisenbaum - April 1964

BIBLIOGRAPHIES (CONTINUED)

"System Requirements for Multiple Access, Time-Shared Computers" - Project MAC, Massachusetts Institute of Technology, Cambridge, Mass - By: F. J. Corbato

"Computer Augmentation of Human Reasoning" - Edited by: Margo A. Sass, Office of Naval Research, and William D. Wilkinson, Bunker-Ramo Corporation, Canoga Park, California

"The TRW Two-Station, On-Line Scientific Computer" - G. J. Culler and B. D. Fried, TRW Space Technology Laboratories, Redondo Beach, Calif.

"An On-Line Computing Center for Scientific Problems M19-3U3" - Glen J. Culler and Burton D. Fried, TRW Computer Division, Thompson Ramo Wooldridge Inc. Canoga Park, Calif.

"Research on Computer-Augmented Information Management" - D. C. Engelbart, Bonnie Huddart, Directorate of Computers, Electronic Systems Division, Air Force Systems Command, USAF, L. G. Hanscom Field, Bedford, Mass.

"User's Guide - Man-Machine Information System" - Stanford Research Institute, Menlo Park, Calif.

"An Experimental On-Line Data Storage and Retrieval System" - J. F. Nolan, A. W. Armenti, Defense Documentation Center, Defense Supply Agency

"Utility of Information as a Predictor of Decision Adequacy in Ambiguous Choice Situations" - HRB-Singer, Inc. Science Park, State College, Pa.

"The MAC System: A Progress Report" - Project MAC, by: R. M. Fano, Massachusetts Institute of Technology, Cambridge, Mass. October 64

"A New Methodology for Computer Simulation" - Project MAC, by: Martin Greenberger, Massachusetts Institute of Technology, Cambridge, Mass. Oct. 64

"Program Structure in a Multi-Access Computer" - Project MAC, by: J. B. Dennis, Massachusetts Intitute of Technology, Cambridge, Mass.

"SIR: A Computer Program for Semantic Information Retrieval" - by: Bertram Raphael, Project MAC, Massachusetts Institute of Technology, Cambridge, Mass. April 64

"The OPS - 1 Manual" - Project MAC, by: Martin Greenberger, Massachusetts Institute of Technology, Cambridge, Mass. May 64

A P P E N D I X

APPENDIX A

Basic Programming System

	Page
I. Description	A1
A. General Nature of the System	A1
B. Description of the Pseudo Computer	A2
II. Operation of the System	A6
A. Control Functions	A6
B. Input Output Functions	A10
C. Storage Areas	A15
D. Arithmetic Operations	A16
1) Fixed Point	A16
2) Floating Point	A21
E. Branching Operations	A25
F. Indirect Programming Operations	A27
G. Miscellaneous Operations	A27
H. Other System Features	A29
1) Tape Access Control	A29
2) Debugging	A29

APPENDIX A

BASIC PROGRAMMING SYSTEM

I. DESCRIPTION

A. General Nature of the System

This appendix describes a programming system developed for use on the BR 85 console as an on line input device to a BR 133 computer. One communicates with the BR 85 through a matrix of thirty buttons. Although only two matrices are physically available to the operator at any one time, the machine has the ability to retain internally 64 different matrices, with the operator having the ability to change at will the matrices in effect on the console.

These matrices are referred to as overlays and are numbered octally from 00 to 77. The basic programming system reserves eight of these overlays, 00 to 07, to provide a language for programming the remaining 56 overlays, 10 to 77. Within the basic system some buttons represent registers. Other buttons represent memory blocks, programmable operations such as add, subtract or transfer. Still other buttons represent switches controlling the operation mode of the computer. Thus the basic overlay system can be described as a pseudo-computer, where several of the basic system buttons represent the switches on the computer, and the remainder of the buttons represent keys on a keyboard for entering pseudo-machine language instructions into the pseudo-computer.

The basic system programming system has within it such capabilities as floating point arithmetic, fixed point arithmetic, and hardware control functions such as magnetic tape control.

The computer has two modes of operation - assemble mode and interpretive mode. When in the assemble mode the computer is utilized to write programs for available keys and to assign other keys as storage area. When writing a program for a key, one is essentially defining a single key to be a sequence of other key pushes. This sequence may contain both basic system keys and other previously defined keys. The computer is placed in the assemble mode by depressing either the DEFINE (0513) or the ASSIGN (0525) key in order to initiate respectively either the writing of a program or the definition of a storage area.

In the interpretive mode, the computer is utilized to execute immediately any and all defined keys. The computer is placed in the execute mode by depressing the EXECUTE (0532) key.

B. Description of the Pseudo Computer

Within the basic programming system which defines the pseudo computer are the following registers. These registers are programmable through buttons on the 00 through 07 overlay levels:

a. Accumulator - fixed point or character

b. Accumulator - floating point

c. Index Registers - 9 each (labelled 1 thru 9)

d. Comparator Register - 1 bit

e. Overflow Indicator - 1 bit

f. Cursor X Register - contains X co-ordinates of display console cursor

g. Cursor Y Register - contains Y co-ordinates of display console cursor

h. Light Gun Address Register - contains address of character last light gunned at the console

i. Special Register - a register for use in indirect programming through which the computer can be directed to execute the numbered button corresponding to the content of this register.

J. Mag Tape Address Register

The following registers are integral to the system but not directly available to the programmer:

K. Instruction Counter Register

L. Secondary Floating Point Accumulator

M. Next Available Core Address Register

N. Current Tape Position Register

O. State Counter Register

In addition to the preceding registers the programmer has at his command two 80-character (word) "buffers":

A. The mag tape buffer thru which all peripheral I/O except console communication occurs

B. The display or digital module buffer which is used for transfer of data to and from the display console's memory unit. (Note: Cursor and light gun do not use this buffer).

Two modes of arithmetic are provided, floating point and fixed point. It is intended that the fixed point be used for calculations such as index manipulation, character comparison and generation, etc. The fixed point arithmetic is integer type, 14-bit + sign (modulo 16,384). Arithmetic is performed with true sign control. However, when an overflow condition occurs the sign will be opposite (overflow occurs into the sign position) and the fixed point overflow indicator will be set. Note that only add, subtract, and multiply can cause overflow. Divide is truncated (i.e., if a fraction is generated it is dropped - no rounding occurs).

Floating point arithmetic is carried out with 29 bits significance and exponent ranging from -2^{1024} to 2^{1024}. Detached sign convention is used. It is necessary during certain move instructions for the programmer to realize that <u>floating point numbers are carried as 3 sequential fixed point numbers</u>. Conventional sign control is performed. Overflow conditions set the floating point overflow indicator. Underflow results in zero.

Control of the computer at the console level is maintained through the use of the overlay keys. The first two keys of every overlay have been dedicated to a specific purpose. Key 01 is the "Increment Overlay Signature" and key 02 is the "Decrement Overlay Signature". Overlays are physically numbered by punching holes along the top border of the template. This gives each specific overlay a "basic" overlay number or "signature". Because it is frequently desirable to work with more than the 60 keys available with two overlays the "increment-decrement system" has been devised to allow a method of effectively changing overlays without physically doing so, thus saving much time. To relieve the console programmer or user of keeping track of how far he has "incremented" from the basic overlay a group of the console status lights has been reserved as an increment indicator. When a new overlay is placed on the console the computer is informed and sets its internal counters properly. Also, one of the "Basic Overlay in Effect" lights is lit. If it was the left overlay placed on the console, it is the "left" light and similarly, if it was the right overlay inserted the "right" light is lit. Now, at any time when the "Increment Overlay Signature" key of the left overlay is depressed, the computer takes note of this fact and henceforth, will add one to the overlay portion of the signature when any other left-hand keyboard key is depressed. Also, the light labelled "Basic Left-Hand Overlay Number +1" is lit and the "Basic Left-Hand Overlay Number in Effect" light is distinguished. As the "increment overlay signature" key is depressed again and again, the lights advance thru "...+2", "...+3", etc. When the "Increment" key is depressed the fifth time the "Left-Hand Overlay Number +5 Plus Above" light is lit, the "Basic Left-Hand Overlay Number in Effect" is lit and the "Basic Left-Hand Overlay Number +4" is extinguished. As the increment key is depressed again and again the light advances as previously thru "...+1", "...+2", "...+3", "...+4". When the increment key is depressed the tenth time, all lights referring to the left-hand overlay are lit plus the "Left-Hand Overlay Number out of Range" light. The computer continues to count and operate correctly. The only differing result is that the status lights are no longer indicating the increment count. The "Decrement Overlay Signature" key is similar to the increment except it subtracts one from the computer's count of the overlay number change and causes the indicating lights to retreat one from whatever position they were in, e.g., from "...+4", to "...+3", etc. If the decrement key is depressed when the computer is indicating "Basic Overlay in Effect" the "Left-Hand Overlay Number out of Range" light will be the only light lit pertaining to the left-hand overlay. As with an increment out of range the computer continues to count, the only difference being indication of the decrement count is not given. The right-hand keyboard functions similarly to the left-hand keyboard, using the right-hand lights instead of the left-hand lights. Should it be desirable to disable the light indicators toggle switch 14 on the computer may be lowered. The only effect is to no longer indicate the overlay differential count. Operation continues the same as before with the computer correctly keeping count of the times the increment-decrement keys are depressed.

II. Operation of the System

A. Control Functions

The following is a description of those keys which can be regarded as switches on the pseudo computer.

Stop key - 0507 (Key number 07 (octal) on overlay 05 (octal))

This key changes certain internal functions of the system that cause the assembler to ignore all communication from the console except the "Start" key (0506). This key is provided to allow the console operator to perform off-line console functions without the computer responding.

Start key - 0506

This key resets all internal functions of the system to the states that existed before the stop key was depressed.

Carry key - 0520

This key causes two subsequent keys to have a synonomous meaning. The next key that is depressed (which must be undefined) is given the meaning of the next key that is depressed. Thus:

"Carry definition to key from key".

Undefine key - 0526

Occasionally it is convenient to unassign a key which has been given a meaning previously. The undefine key provides this function. Press "Undefine" and then the key to be undefined. Certain "lock-out" protection features are incorporated in the function of this key. One, no key which is a system provided key may be undefined and two, only keys which are located on an overlay which is two or more overlays higher than the overlay inserted in the console may be undefined. Undefined keys may be undefined providing the two or higher rule is followed.

Execute - 0532

This key when depressed places the system in the interpretive mode. The next key depressed causes the system to execute the program defined for that key. The execute mode is terminated whenever a key is depressed which causes an assembler function to be initiated.

Print Program - 0413

This key causes the program (or assignment) of the next key depressed to be printed on the typewriter in the octal overlay-key format suitable for reentry to the system at a later date.

Punch Program - 0414

Same as 0413 except Punch Paper Tape instead of type.

Paint and Punch Program - 0415

Same as 0413 except Punch Paper Tape and Type.

Line Print Program - 0416

Same as 0413 except Line Print instead of type.

Punch and Line Print Program - 0417

Same as 0413 except Line Print and Punch instead of type.

Display Program - 0420

Same as 0413 except Display on Crt instead of type.

Purge - 0021

This key removes all useless information from the working magnetic tape. This key should probably be used after a session of console work and any other time after many "un-assignments" have been accomplished or considerable usage of different programs. Frequency of use can only be determined by experience. After purging the computer will stop. To restart press the "flag" button on the computer control panel (130) or the "start" switch on the computer control panel (133).

Assign Storage - 0525

This key sets the assembler to a state that allows the console programmer to specify a key that will henceforth have the meaning of a data storage area. After the "Assign Storage" key is depressed, the assembler expects the key that is to be assigned to be depressed. This key must be undefined. After the key to be assigned is pressed, the assembler must be given the dimension of the storage area being defined. This is done by pressing the number keys on overlay 05. After the proper numbers keys have been pressed, the assembler is so informed by the console programmer pressing the key labeled "/ Stop Code" 0536. Finally, the End key - 0514 must be pressed. Thus, if one wishes for key 27 of overlay 11 to represent an 80-word storage area the following keys would be depressed in the following order:

Assign Storage - 0525, Key being defined - 1127, Number 8 - 0516, number 0-0535, / Stop Code - 0536, End - 0514.

Note: Storage keys may be used in the assembling of a program before they are assigned. However, if a program, using an unassigned storage key is executed an interpretive error will occur.

Note: An additional feature is provided whereby the programmer may specify the contents of each word of the assigned area. This is accomplished just as one would insert constants into a program (see keys 0503-0504-0505) prior to pressing the end -0514- key. If all words of the operand are not specified in this manner, the remaining words will be set to the value of the last specified word If no contents are specified then zero is stored throughout the operand.

Define - 0513

The define key starts the assembly of a console program. The next key depressed will assume the title of the program that is being defined. Whenever the "title" key is subsequently depressed it will call for its program. During the ensuing discussion of the various operation and operand keys the console programmer must keep it clearly in mind that the key definition is being assembled into a program which will not be executed until after assembly is complete. That is, the operation and operand keys cause no interpretive action at the time of assembly but simply go into a list for execution at a later time. For most operations it is more convenient to explain the action caused at execute time and thus the reason for inclusion of an operation in the program.

End - 0514

This is always the last key depressed when DEFINEing a program or ASSIGNing an operand. It instructs the computer that the program definition of storage assignment has been completed.

B. Input Output Functions

The following instructions are programmable and control the input-output devices.

Read Mag Tape - 0603

This key causes (at execute time) a mag tape record (normally 80 characters) to be read and stored in the mag tape buffer. Access to the information in the mag tape buffer is accomplished with various operation keys to be described later. Should an EOF (End of file) mark be read from tape the fixed point overflow indicator will be set. If a record is read the fixed point overflow indicator will be turned off (Unset).

Read Mag Tape EOF - 0604

This key causes (at execute time) the mag tape to be moved forward until an EOF (End of file) mark is sensed. The tape is then in position to read the record following the EOF.

Rewind Mag Tape - 0605

This key causes the mag tape to rewind to its "load point" (beginning of tape).

Write Mag Tape - 0610

This key causes the information in the mag tape buffer (normally 80 characters) to be written onto the mag tape.

Write Mag Tape EOF - 0611

When this key is executed an EOF (End of file) mark is recorded on the mag tape.

Backspace Mag Tape - 0612

The mag tape is backed up one record length when this key is executed.

Mag Tape Register - 0615

This is a special register used for mag tape designation. Normally the mag tape manipulation is set up for controller 1 and transport 1. Data is handled in Alpha format. These specifications may be changed under program control by storing a different pattern into this register. The octal coding of the register is as follows:

5	4	3	2	1	
0					for alpha
4					for binary
	0	0			must be zero
			X		controller number 0, 1, 2, 3
					controller 4 is specified by 0
				X	transport number 0, 1, 2, 3
					transport 4 is specified by 0
0	0	0	1	1	normal set up - alpha-controller 1 - transport 1

Read Typer - 0625

This key causes the typewriter keyboard (on 141 or 143) to be connected as an input device. Characters are read from the typewriter as the keys are actuated until either 80 characters and/or functions have been entered or until a carriage return is executed. The characters and/or functions are stored in the mag tape buffer starting at its beginning position and continuing one character or function per element in the buffer. If fewer than 80 entries are made the remaining portion of the buffer is set to space characters.

Write Typer - 0626

This key causes the contents of the mag tape buffer to be output on the typewriter. At any time during this operation that it is determined that the unoutput portion of the mag tape buffer contains all space characters or that 80 characters and/or functions have been output the typewriter is caused to carriage return and the operation is terminated.

Read Paper Tape - 0630

This key is similar to the read typer - 0625 except the paper tape reader is used as the input device.

Punch Paper Tape - 0631

This key is similar to the writer type - 0626 except the paper tape punch is used as the output device.

Read Card - 0632

This key causes a card to be read from the card reader and the 80 columns of Hollerith coding to be stored as characters (in the proper code set) in the mag tape buffer.

Write Line Printer - 0627

This key, when executed, causes the contents of the specified "operand" (see later description of "block operands") to be output on the line printer. The line printer will accommodate 120 characters per line, thus, the operand may be as long as 120 words. When printing from the operand the least significant 6 bits only, of each word, are translated to appropriate characters. When the word containing the first character to be output also contains a sign bit, the line printer will skip to the first line of the next page before printing. When the sign bit of the first word is not present, the line printer advances to the next line and prints. The next line after the last line of a page is considered to be the first line of the next page. Thus at page ends there is about a 1" space of unprinted space.

Read Toggles - 0634

This key causes the left six computer toggle switch settings to be transferred to the least significant 6 bits of the fixed point (character) accumulator. A one bit is represented by the toggle being up. A zero is indicated by the toggle being down.

Write Display - 0636

This key, when executed, causes the contents of the display module buffer to be transferred to the display console CRT. Because the display module memory is addressable the system must supply beginning and ending addresses of where the data is to be stored. This is accomplished by placing the beginning address in the 79th position of the display buffer and the ending address in the 78th position. The ending address may be less than 78 greater than the beginning address, in which case only a portion of the display buffer will be output to the console. In the case of the ending address being exactly 78 greater than the beginning address the entire buffer contents will be output. If the ending address is more than 78 greater than the beginning address the contents of the display buffer will be completely output and the operation terminated; i.e., never will more than the complete buffer be output and no indication is given if more is called for. Never write in addresses 00 through 17 (octal).

Read Display - 0635

The key causes the display module's memory to be read into the display buffer. As in the case of "Write Display" the beginning and ending addresses, to be read from and to, must be placed in the display buffer elements 79 and 78 respectively. Never read from addresses 00 through 17 (octal).

C. Storage Areas

Storage areas may be defined as explained under the "Assign Storage" key - 0525. The assignment discussion implies that the size (number of words) of a storage area may be controlled by the console programmer. Because it is frequently convenient to use only a portion of a storage area for a particular function, provision has been made in the system to allow this. It is possible to specify not only the element of the area to be considered, but also the number of elements.

The beginning element may be specified in a number of ways. Perhaps the easiest method is by the use of a constant. This is entered into the program immediately after the operand key is pressed by pressing the "Decimal" key - 0504 and then a sequence of the "numeric" keys - 0535 - 0527 through 0532 - 0522 through 0523 - and 0515 through 0517 - followed by the "stop code" key - 0536. Note that the first element of a storage area is referred to as the "zero" element or word. The alternate method of specifying the first element to use is by the use of one of the index registers or the fixed point accumulator. Before execution of an operation needing an indexed operand the contents of the index register must be "set" to a meaningful value. Upon execution of the operation the contents of the index register are taken to mean the beginning element of the storage area to be used. In any case, where the element value exceeds the dimension of the storage area an interpretive error will result at execution time.

As mentioned above, the number of elements to be used may be specified as well as the first element to use. If no limit is imposed then it is assumed all elements to the end of the storage area are to be used. A limit may be imposed by inserting a constant (similar to the first element constant) after the first element description. Also, any index register or the fixed point accumulator may be specified as containing the count of the number of elements to use. (Processing one element is specified by one not zero).

In the case where two storage areas are used (such as in a block transfer) termination of the operation will occur whenever the end element of either area is processed or when the smallest limit value (if any) has been exhausted, whichever occurs first; i.e., it is impossible to process beyond the end of a storage area.

D. Arithmetic Operation

1) Fixed Point

Transfer Non-Block - 0606

This transfer operation requires two single word operands. These operands may be fixed point registers (index registers, fixed point accumulator), assigned storage operands with a dimension of one, assigned storage operands with a dimension greater than one and an element index specified or the mag tape or digital module buffer with an element index specified. The operation is to transmit the contents of the first specified operand into the second operand. Note that this operation is both load and store, being defined by the order of the operands and also that data may be transmitted between assigned storage elements without the need of loading the accumulator.

Exchange Non-Block - 0607

This operation is similar to the transfer operation above except that data is transmitted both ways; i.e., an interchange of the contents of the two specified operands occurs.

2's Complement - 0130

Convert the contents of the fixed point accumulator to 2's complement form and store back in the character accumulator (0 minus FXACC to FXACC).

1's Complement - 0131

Convert the contents of the fixed point accumulator to 1's complement form and store back in the fixed point accumulator (reverse each bit).

Extract - 0135

Perform a logical "or" with each bit of the specified operand and the corresponding bit of the fixed point accumulator, leave the result in the fixed point accumulator. Example:

Acc Start	1100
Operand	0101
Acc Result	0100

Add - 0106

Add the contents of the specified operand to the fixed point accumulator. If overflow results set the fixed point overflow indicator on.

Subtract - 0107

Subtract the contents of the specified operand from the fixed point accumulator. If overflow results set the fixed point overflow indicator on.

Multiply - 0113

Multiply the contents of the fixed point accumulator by the contents of the specified operand. Place the product in the fixed point accumulator. If overflow results set the fixed point overflow indicator on.

Divide - 0114

Divide the fixed point accumulator by the contents of the specified operand. Truncate the quotient placing the integer portion in the fixed point accumulator.

Increment X - 0123

This operation must be followed by one of the nine index registers. The operation adds one to the specified index register's contents.

Decrement X - 0124

Subtract one from the contents of the specified index register.

BCD to FXACC - 0035

The operand for this operand specifies the beginning of a sequence of words containing decimal numbers. The operation is to binarize the string of numbers and place the result in the fixed point accumulator. The first valid character may be preceded by spaces which are ignored. A sign + or - may be anywhere in the string of numbers. If more than one sign is present the last one encountered will control the sign of the result. No sign is taken to mean positive. Binarization will continue until a non-numeric, non-sign character or until the end of the operand containing the string is encountered. If overflow results the fixed point overflow indicator will be set.

FXACC to BCD - 0036

The operand for this operation specifies the beginning of a sequence of words. The operation is to debinarize (convert to a string of BCD characters) the contents of the fixed point accumulator and store the result in the specified string of words. The sign of the accumulator is stored in the first word of the string (- for minus, space for plus). The most significant character of the number is stored in the next word, the next number in the next word, etc., until up to five characters have been stored. If the specified string has remaining words in it, they are filled with spaces. Whenever the end of the string is encountered the operation is terminated regardless of whether the complete debinarized number string has been stored. No provision is made to indicate the fact, therefore, it is the responsibility of the console programmer to provide a large enough string for the number to be debinarized (six is always sufficient).

Compare Fixed Greater Than - 0125

The contents of the specified operand are compared with the fixed point accumulator. If the operand is greater than the accumulator; the comparator register is set true. If the operand is equal to or less than the accumulator; the comparator register is set false.

Compare Fixed Equal to - 0126

The contents of the specified operand are compared with the fixed point accumulator. If the operand is equal to the accumulator; the comparator register is set true. If the operand is not equal to the accumulator; the comparator register is set false.

Compare Fixed Less Than - 0127

The contents of the specified operand is compared with the fixed point accumulator. If the operand is less than the accumulator; the comparator register is set true. If the operand is equal to or greater than the accumulator; the comparator register is set false.

Set True - 0030

Set the comparator register true (no operand needed).

Set False - 0031

Set the comparator register false (no operand needed).

2) Floating Point

Transfer Block - 0624

This transfer operation requires two multiple word operands. These operands may be the floating point accumulator, assigned storage operands with a dimension greater than one or the mag tape or digital module buffer. In any case except the floating point accumulator a first element specifier is required. This may be in the form of a constant or index register. Optionally an "n" number of elements to be transferred may be given with either or both operand following the first element specifier. The block transfer will continue until either of the operand's last element has been processed (from or to) or until "n" number of elements has been processed, whichever is sooner.

It is interesting to note that the transfer is processed in ascending order of the elements - thus, if the first element of a block is specified to be transferred to the second element of the same block and no "n" is given, then the entire block will be set to the value in the first element. Conversely, if the second element of a block is specified to be moved to the first element of the same block the effect is to "ripple" the contents of the entire block down one element. The last element and the next to last element will contain identical values.

When transferring floating point data the block transfer mode should be employed. If the optional "n" value is used it must be three times the number of floating point numbers to move.

Fixed Point to Floating and Floating Point to Fixed

To accomplish the operations of "float" and "unfloat" a special connotation has been given to the Transfer-Non-Block - 0606 operation when the floating point accumulator is involved.

If a non-block transfer is made to the floating accumulator, the operand is considered to be fixed point, is "floated" and placed in the floating accumulator.

If a non-block transfer is made from the floating accumulator, the floating accumulator is "unfloated" and the result stored in the operand. Should overflow occur during the "unfloat" operation the fixed point overflow accumulator is set for on and the largest possible positive number is stored in the operand. Underflow causes zero to be stored in the operand.

Add - 0006

Add the contents of the specified operand to the floating point accumulator. If overflow results set the floating point overflow indicator on.

Subtract - 0007

Subtract contents of the specified operand from the floating point accumulator. If overflow results set the floating point overflow indicator on.

Multiply - 0013

Multiply the floating point accumulator by the contents of the specified operand.

Divide - 0014

Divide the floating point accumulator by the contents of the specified operand.

Sin - 0010

Compute the sine of the floating point accumulator (in radians) and store the result in the floating accumulator. No operand required.

Cos - 0015

Compute the cosine of the floating point accumulator (in radians) and store result in the floating accumulator. No operand required.

Atan - 0011

Compute the arctangent of the floating point accumulator and store the result (in radians) in the floating accumulator. (No operand required).

Sqrt - 0016

Compute the square root of the floating accumulator and store the result in the floating accumulator. If the contents of the accumulator was negative - terminate the operation and execute the "executive error" function.

Log - 0012

Computer the logarithm to base 10 and store the result in the floating accumulator. If the argument was zero or negative terminate the operation and execute the "executive error" functions.

Exp - 0017

Compute the anti-logarithm to base 10 and store the result in the floating accumulator.

Compare Floating Greater than or Equal - 0025

The contents of the specified operand are compared with the floating accumulator. If the operand is greater than or equal to the accumulator; the comparator register is set true. If the operand is less than the accumulator; the comparator register is set false.

Compare Floating Less than - 0027

The contents of the specified operand are compared with the floating accumulator. If the operand is less than the accumulator; the comparator register is set true. If the operand is greater than or equal to the accumulator; the comparator register is set false.

Change Sign Floating - 0023

Change the sign of the floating accumulator. No operand required.

Absolute Floating - 0024

Make sign of the floating accumulator positive. No operand required.

E. Branching Operations

Ten "Jump" or branch functions are provided. When in the assemble mode the next key depressed after a jump or branch function is not interpreted to mean assemble that key. Instead, this key is interpreted as a label or reference point in the program. For convenience each label is identified by its corresponding button number. A Jump operation is always followed by a "label" designating where to jump to. A "label" may be <u>any</u> key on any overlay except the "increment" and "decrement" keys. (The key used as a "label" may also have another function). Keys used as labels are only effective between a "define" and "end" function; i.e., "define" initializes such that no keys have a "label" meaning. In order to place a "label" in a program for jumps to be meaningful the following key has been implemented.

Label Follows - 0521

This key is always succeeded by a "label" key in order to establish a reference point. The effect is to give previous or later jump operations a destination. That destination is the operation immediately following the "label key" after a "label follows" function. The "label key" must conform exactly to the key used after the appropriate jump, i.e., overlays must match.

Jump True - 0132

If the comparator register is true - jump to the labelled operation. Otherwise, execute the next sequential operation.

Jump False - 0133

If the comparator register is false - jump.

Jump Floating Plus - 0020

If the floating point accumulator is positive - jump.

Jump Floating Minus - 0022

If the floating point accumulator is negative - jump.

Jump Fix Plus - 0120

If the fixed point accumulator is positive - jump.

Jump Fix Zero - 0121

If the fixed point accumulator is zero - jump.

Jump Fix Minus - 0122

If the fixed point accumulator is negative - jump.

Jump Floating Overflow - 0032

If the floating point overflow indicator is on - set it off and jump. Otherwise execute the next sequention operation.

Jump Fixed Overflow - 0033

If the fixed point overflow indicator is on - set it off and jump.

Jump Always - 0134

Always jump.

F. Indirect Programming Operations

Carry at Execute - 0026

This operation must be followed by two keys. These keys must be of the same format, i.e., either storage areas or executable programs, operations or functions. The decimal keys may not be used for this purpose. The effect of the operation is to make the second specified key take on the meaning of the first specified key. This key is programmable and should not be confused with the carry key. The carry key is a switch on the computer for immediately accomplishing the same task.

Read Program Key - 0623

This key causes the computer to go into an idle mode waiting for a program key to be pressed. When the key is pressed its signature (in octal notation) is placed in the special register defined by key 0003 and 0533.

Substitute - 0533

This key is replaced by the key whose signature is in the special register (see 0003 and 0623). Thus, with the use of Read Program Key - 0623 an operand or operation may be dynamically specified at execution time.

Special Register - 0003

The contents of this register may be loaded with a non-block transfer - 0606 or by Read Program Key - 0623. The important use of this register is with the substitute - 0533 function. Thus register is addressable from the fixed accumulator and as such can be utilized in executing a calculated key number.

G. Miscellaneous Operations

Abort - 0411

This key when executed causes the computer to return to idle mode waiting for another key to be pressed for execution. This is useful when an error is diagnosed in a console program and no recovery is possible.

Zeroize operand - 0412

Store a zero into every word of the specified operand. No element index or optional "n" is allowed.

Break point 1 - 0403

If toggle 1 on the computer is up then stop the computer. Press start to restart and execute next sequential key. If toggle 1 on the computer is down then execute the next sequential key. This feature is useful for console program debugging.

Break point 2 - 0404

Same as Break point 1 except use computer toggle 2.

Break point 3 - 0405

Same as Break point 1 except use computer toggle 3.

Break point 4 - 0406

Same as Break point 1 except use computer toggle 4.

Break point 5 - 0407

Same as Break point 1 except use computer toggle 5.

Break point 6 - 0410

Same as Break point 1 except use computer toggle 6.

Call Program or operand - 0421

This key is used for optimization of console program.

H. Other System Features

1. Tape Access Control

Console programs and operands are called into core from mag tape when they are first required. They stay in core until the program calling them has been finished. In order to prevent the mag tape from "dancing" to get programs and operands in the order needed this "call" key will allow programs and operands known to be needed to be called in the order they appear on mag tape. Each depression of the "call" key requires a key indicating a console program or console assignment to follow. No execution of the "called" program or operand is implied or need be done.

2. Debugging

If computer toggle switch 11 is up the computer will stop before executing each console operation. Also the computer may be stopped by the use of Break point keys and corresponding computer toggle switches.

When the computer is stopped for the above reasons the signature of the key to be executed is displayed in the least significant 12 bits of the "A" register. The comparator register is displayed in bit 15 of the "A" register. (On = 1 or true, off = 0 or false).

The contents of the fixed point accumulator is displayed in the "P" register. The "T" register contains in bit 15 the sign of the floating accumulator, in bits 6 thru 1 the 2's exponent of the floating accumulator and in bits 14 thru the most significant part of the floating accumulator in positive form. The

"L" register contains (in binary) the program location of the key stopped on. To restart the computer in order to execute the next operation (or the one being displayed in the case of switch 11 being up) press the computer start switch. To abort the entire operation and return to the "idle mode" place the computer in manual operation - press "display" switch - place the computer in normal mode and press the start switch.

Synopsis

Overlay	Key	Title	Remarks
any	01	Increment	Increase the overlay signature in effect by 1
any	02	Decrement	Decrease the overlay signature in effect by 1
00	03	Special Register	Used in conjunction with Read program key and substitute
00	04	BCD - FLACC	Binarize and Float String to Fl Acc
00	05	FLACC - BCP	Unfloat and Debinarize Fl Acc to String
00	06	Floating ADD	
00	07	Floating SUB	
00	10	Floating SIN	Radians
00	11	Floating ATN	Radians
00	12	Floating LOG	Base 10
00	13	Floating MUL	
00	14	Floating DIV	
00	15	Floating COS	Radians
00	16	Floating Square Root	
00	17	Floating EXP	Base 10 anti-log
00	20	Floating Jump positive	
00	21	Purge	Remove discarded records from mag tape press computer start to continue after purge
00	22	Floating Jump Negative	
00	23	Floating Change Sign	
00	24	Floating Absolute	
00	25	Compare Floating Greater than	Set comparator true if operand is greater than or equal to the floating accumulator

Overlay	Key	Title	Remarks
00	26	Carry At Execute	Carry to key from key
00	27	Compare Floating Less Than	Set comparator true if operand is less than the floating accumulator
00	30	Set True	Set comparator true
00	31	Set False	Set comparator false
00	32	Jump Floating overflow	
00	33	Jump Fixed overflow	
00	34	Vacant	
00	35	BCD - FXACC	Binarize String to Fixed point accumulator
00	36	FXACC - BCD	Debinarize Fixed point accumulator
01	01	Increment	Increase the overlay signature in effect by 1
01	02	Decrement	Decrease the overlay signature in effect by 1
01	03	Index 9	
01	04	Index 8	
01	05	Index 7	
01	06	Fixed Add	
01	07	Fixed Sub	
01	10	Index 6	
01	11	Index 5	
01	12	Index 4	
01	13	Fixed Mul	
01	14	Fixed Div	
01	15	Index 3	

Overlay	Key	Title	Remarks
01	16	Index 2	
01	17	Index 1	
01	20	Jump Fixed positive	
01	21	Jump Fixed zero	
01	22	Jump Fixed Negative	
01	23	Increment Index	Add 1 to specified index register
01	24	Decrement Index	Subtract 1 from specified index register
01	25	Compare Fixed greater	Set comparator true if operand greater than accumulator
01	26	Compare Fixed Equal	Set comparator true if operand equal to accumulator
01	27	Compare Fixed Less	Set comparator true if operand less than accumulator
01	30	2's Complement	Multiply accumulator by minus 1
01	31	1's Complement	
01	32	Jump true	
01	33	Jump false	
01	34	Jump always	
01	35	Extract	Bit by bit and of operand and accumulator
01	36	Vacant	
02	01	Increment	Increase the overlay signature in effect by 1
02	02	Decrement	Decrease the overlay signature in effect by 1
02	03	A	For use in conjunction with ALF key 0503

Overlay	Key	Title	Remarks
02	04	B	
02	05	C	
02	06	D	
02	07	E	
02	10	F	
02	11	G	
02	12	H	
02	13	I	
02	14	J	
02	15	K	
02	16	L	
02	17	M	
02	20	N	
02	21	O	
02	22	P	
02	23	Q	
02	24	R	
02	25	S	
02	26	T	
02	27	U	
02	30	V	
02	31	W	
02	32	X	
02	33	Y	
02	34	Z	
02	35	STOP CODE	

Overlay	Key	Title	Remarks
02	36	SPACE	
03	01	Increment	Increase the overlay signature in effect by 1
03	02	Decrement	Decrease the overlay signature in effect by 1
03	03	--	For use in conjunction with ALF key 0503
03	04	$	
03	05	↓	Marker
03	06	.	Period
03	07	|	Straight line
03	10	Stop Code	
03	11	/	slash
03	12	#	Number sign
03	13	,	comma
03	14	(	Left parenthesis
03	15	)	Right parenthesis
03	16	$\leq$	Epsilon
03	17	Δ	Delta
03	20	-	Minus sign
03	21	+	plus sign
03	22	±	plus-minus sign
03	23	*	asterisk
03	24	"	Quotes
03	25	°	degree mark
03	26	>	greater than mark

Overlay	Key	Title	Remarks
03	27	%	Per cent
03	30	<	less than mark
03	31	'	apostrophe
03	32	____	under line
03	33	:	Colon
03	34	?	Question mark
03	35	•	Plot symbol
03	36	Vacant	
04	01	Increment	Increase the signature of the overlay in effect by 1
04	02	Decrement	Decrease the signature of the overlay in effect by 1
04	03	BP 1	Break point 1
04	04	BP 2	Break point 2
04	05	BP 3	Break point 3
04	06	BP 4	Break point 4
04	07	BP 5	Break point 5
04	10	BP 6	Break point 6
04	11	ABORT	Terminate program being executed
04	12	Zeroize operand	Store zero throughout specified operand - no element index or optional "n" allowed
04	13	Print program	
04	14	Punch program	
04	15	Print and Punch program	
04	16	Line Print program	

Overlay	Key	Title	Remarks
04	17	Punch and Line print program	
04	20	Display program	
04	21	Call program or operand	
04	22	Binary Dump	
04	23	2K operand	
04	24	Punch 507	
04	25	Executable stop	
04	26	Revised paper tape	
04	27	Assemble from 2K operand	
04	30	Vacant	
04	31	Vacant	
04	32	Vacant	
04	33	Vacant	
04	34	Vacant	
04	35	Vacant	
04	36	Reserved	
05	01	Increment	Increase overlay signature in effect by 1
05	02	Decrement	Decrease overlay signature in effect by 1
05	03	Alphabetic	Prepare to insert alpha constant
05	04	Decimal	Prepare to insert decimal constant
05	05	Octal	Prepare to insert octal constant
05	06	Start	Reinitiate action after Stop 0507

Overlay	Key	Title	Remarks
05	07	**Stop**	Ignore all keys and light gun except Start 0506
05	10	Special Register	May be used for a computed assign dimension
05	11	+	Make the constant being inserted positive
05	12	-	Make the constant being inserted negative
05	13	Define	Initiate program definition
05	14	End	End assembly of a program or assignment for an operand
05	15	9	For use in conjunction with ALF or Dec, 0503-0504
05	16	8	For use in conjunction with ALF or Dec, 0503-0504
05	17	7	For use in conjunction with ALF or Dec, or OCT, 0503-0504-0505
05	20	Carry	Carry meaning to key from key
05	21	label follow	The next key is a label
05	22	6	For use in conjunction with ALF-Dec or OCT, 0503-0504-0505
05	23	5	For use in conjunction with ALF-Dec or OCT, 0503-0504-0505
05	24	4	For use in conjunction with ALF-Dec or OCT, 0503-0504-0505
05	25	Assign	Initiate assignment of an operand
05	26	Undefine	Undefine the next key - must be 2 overlay levels above
05	27	3	For use in conjunction with ALF-Dec or OCT, 0503-0504-0505
05	30	2	For use in conjunction with ALF-Dec or OCT, 0503-0504-0505
05	31	1	For use in conjunction with ALF-Dec or OCT, 0503-0504-0505

Overlay	Key	Title	Remarks
05	32	Execute	Prepare to execute console programs
05	33	Substitute	Use the key whose signature is in the special register
05	34	Infinity	For use in conjunction with ALF-Dec , OCT, 0503-0504-0505
05	35	zero	For use in conjunction with ALF-Dec, OCT, 0503-0504-0505
05	36	Stop Code	Terminates insertion of constant initiated by ALF-Dec -OCT, 0503-0504-0505
06	01	Increment	Increase the overlay signature in effect by 1
06	02	Decrement	Decrease the overlay signature in effect by 1
06	03	Read MT	Read a record from mag tape into the mag tape buffer - 0617. If EOF is read turn on fixed point overflow
06	04	Read MT EOF	Read mag tape records until an EOF is read
06	05	Rewind MT	Rewind the mag tape
06	06	Transfer	Transfer data from the next operand to the next
06	07	Exchange	Exchange data between the two following operands
06	10	Write MT	Write the mag tape buffer - 0617 onto mag tape
06	11	Write MT EOF	Write a mag tape EOF
06	12	Backspace MT	Backspace over one mag tape record
06	13	Floating Accumulator	3 words

Overlay	Key	Title	Remarks
06	14	Fixed point Accumulator	1 word
06	15	Mag tape register	Contents specify Controller - Transport and binary or alpha mode
06	16	Digital Module Buffer	78 = Ending address, 79 = beginning address
06	17	Mag tape Buffer	
06	20	Cursor X Register	Hold cursor X co-ordinates when read cursor 0633 is executed
06	21	Cursor Y Register	Hold cursor Y co-ordinate when read cursor 0633 is executed
06	22	Light gun Address	Holds address of character addressed
06	23	Read Program Key	Idle till key can be read - then sig. to special register
06	24	Block transfer	for transferring Blocks of data from next operand to the next - Floating point is 3 words
06	25	Read typewriter	Read from typewriter for 80 characters or until CR into mag tape buffer
06	26	Write typewriter	Write contents of mag tape buffer on typewriter
06	27	Write Line Printer	Write contents of next operand on line printer
06	30	Read paper tape	Similar to 0625 except for source of data
06	31	Punch paper tape	Punch contents of mag tape buffer into paper tape
06	32	Read Card	Read Hollerith from card into mag tape buffer
06	33	Read Cursor	Store cursor co-ordinates in X and Y registers 0620 and 0621

Overlay	Key	Title	Remarks
06	34	Read Toggles	Computer toggles 1 thru 6 to fixed point accumulator 1 = up 0 = down
06	35	Read Display	Digital Module Memory to Digital Module buffer
06	36	Write Display	Digital Module Buffer to Digital Module Memory

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* The Bunker-Ramo Corporation	2a. REPORT SECURITY CLASSIFICATION Unclassified	
	2b. GROUP	
3. REPORT TITLE On-Line Command and Control Study		
4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)* Research Progress Report May 1963 through November 1964		
5. AUTHOR(S) *(Last name, first name, initial)* Martins, Gary R. Wilkinson, William D.		
6. REPORT DATE September 1965	7a. TOTAL NO. OF PAGES 77	7b. NO. OF REFS 1
8a. CONTRACT OR GRANT NO. Nonr 4182(00)	9a. ORIGINATOR'S REPORT NUMBER(S) D58-5U1	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	
d.		
10. AVAILABILITY/LIMITATION NOTICES "Qualified requesters may obtain copies of this report from DDC."		
11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Office of Naval Research	

13. ABSTRACT

In most automated Command Information Systems, the military user is hampered by the lack of convenient means for communication with the computer system and by the inflexibility of the system to respond to a rapid change either in the type of problem to be solved or in the method of solving a standard problem. It is in these areas that on-line techniques show promise for improvement. The major objective of on-line techniques is to put the user directly in the problem-solving system loop in such a manner that the experience and intuition of the user becomes closely coupled with the powerful computational capabilities of the computer so that there is a balanced interaction between the capabilities of each.

On-Line techniques employ a functional approach to permit the user to:

1) communicate with the computer system conveniently in his own terminology,
2) structure the problem solving process, and
3) maintain continuous control over the system.

Although Command and Control is emphasized in this study, the techniques described are general and should be useful in a wide range of problem solving applications.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
On-Line Information Processing						
Command and Control						
Command Information Systems						
On-Line Systems						
Time-Sharing						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through
_______________________________________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through
_______________________________________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through
_______________________________________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.