NATIONAL LIBRARY
OTTAWA

CANADA

BIBLIOTHÈQUE NATIONALE
OTTAWA

6237

NAME OF AUTHOR...Eugene William Romaniuk

TITLE OF THESIS..A VERSATILE AUTHORING LANGUAGE FOR TEACHERS

UNIVERSITY.....The University of Alberta

DEGREE FOR WHICH THESIS WAS PRESENTED...Ph.D.

YEAR THIS DEGREE GRANTED....1970 (Spring)

Permission is hereby granted to THE NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed). E. W. Romaniuk

PERMANENT ADDRESS:

11242 - 97 Street
Edmonton, Alberta

DATED.....April 8.........1970

THE UNIVERSITY OF ALBERTA

A VERSATILE AUTHORING LANGUAGE FOR TEACHERS

BY

© EUGENE WILLIAM ROMANIUK

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF EDUCATIONAL PSYCHOLOGY

EDMONTON, ALBERTA

SPRING, 1970

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "A Versatile Authoring Language For Teachers" submitted by Eugene William Romaniuk in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

S. Hunka
Supervisor

Willard N. Runquist

Ian MacInnes

Thomas O. Maguire

John J. Schurdak
External Examiner

Date January 16, 1970

ABSTRACT

This study investigated the possibility of producing a new computer authoring language. The new language, named VAULT (a Versatile AUthoring Language for Teachers), was primarily designed for use by teachers who wish to produce materials for computer based instruction.

The major objectives of the study were: (*a*) to design an authoring language (VAULT) which separates logic and subject matter and uses natural terms that are meaningful to the activities of the classroom teacher; (*b*) to develop and implement a subset of VAULT; (*c*) to teach VAULT to a group of teachers and appraise their attitudes toward the language; (*d*) to evaluate the VAULT subset; and (*e*) to draw conclusions, from the results obtained with the subset, regarding the feasibility of developing and implementing the authoring language.

A subset of VAULT was developed. The VAULT compiler, written in PL/1, was implemented on the IBM System 360, Model 67; it accepts VAULT input and produces source code which is then assembled on the IBM 1500 Instructional System. A major portion of the facilities provided by the 1500 System could be controlled by VAULT with the exception of audio and system functions.

The VAULT subset was taught to a group of teachers during the summer of 1969. Tentative results indicate an increased authoring efficiency which would make further investigation and research highly desirable.

ACKNOWLEDGEMENTS

The author wishes to especially thank his supervisor, Dr. S. M. Hunka, for his guidance and encouragement so willingly given throughout the preparation of this thesis. I am also deeply appreciative of the constructive comments and valuable suggestions of my committee members.

In appreciation of their dedication and assistance to the study the author wishes to acknowledge Mr. Reg Jordan and Mrs. Wendy Birtch, IBM systems engineers.

Finally, the author wishes to acknowledge his gratitude to the University of Alberta for providing much needed assistance during the past year and to the International Business Machines Corporation (Canadian Laboratory) for providing grant support for the implementation phase of this project.

DEDICATION

To my wife Camille.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

INTRODUCTION

Since the end of World War II and in particular since the orbiting of the first Sputnik in 1957, there has been a tremendous surge of interest in and commitment to the cause of education in America. The past twenty years has provided us with what can be described as a "knowledge explosion." As a result, this mushrooming of knowledge has begun to change society and in turn the resulting social forces are compelling educational institutions to redefine their functions and processes.

Education, one of the cornerstones of our society, is faced with a number of major challenges. Much attention and publicity has been focused on the impending crisis of our educational resources in coping, expanding, and adapting to changing education and technology (Mitzel, 1967). Schools must show progress and improvement throughout the entire educational system in order to meet the stresses and strains of the dynamic and complex forces acting upon the educational institutions. The public is demanding more and better education; students attend school for more years than they have previously; more adults require job retraining to counter job obsolescence and the increasingly more complex new jobs becoming available as a result of new technologies; the student population has become much larger due to an increase in population; schools must impart more facts and ideas than ever before; students are demanding new approaches and improved methods of teaching; and the cost of education is ever increasing thus causing an increased burden on the taxpayer.

Decision-makers and teachers involved in the educational process are faced with these problems and must seek more efficient and productive ways of operating our schools in a period when educational costs are growing at the fastest rate in history. However, although many segments of the educational institutions are rising to meet the challenge society presents, many schools are still geared to teach traditional matter in much the same manner as it has been done for the last twenty-five years. Due to the economics involved in administration of public schools, we still use, to a large extent, a system in which students of the same age level are grouped together and taught the same information at the same rate.

Recently, students have become increasingly outspoken in their criticism of the education they receive. Their cry has been for a more liberal educational atmosphere in which the student can learn by interacting with his teacher in a "Socratic" climate. This could perhaps be interpreted as a demand for a form of individualized education in which the student has some voice in what he learns.

Schools are slowly beginning to change in response to the fundamental difficulties facing education. The changing concept of education includes an increasing awareness for a wide range of social problems. A new concern is being raised as to who is taught, what is taught, how it is being taught, and how well it is being taught. Generally, the ideal of future education is that since education is a continuing process during the life of the individual, a high quality education should be made available continuously to all persons. Education is faced with the task of finding new teaching methods and technologies to fill the increasing need of an improved education in the rapidly changing schools. One approach that may help alleviate some of the burden is computer based instruction. "A computer can provide completely individualized instruction in which the pace and sequence of materials are independently controlled for each student, based on his responses to the materials [Coulson, 1966, p. 340]."

A history of computer based instruction and a review of relevant research literature are presented in chapter two. Chapter three describes a number of early computer based instructional systems and, as well, the hardware and software features of current major computer based instruction systems.

Three aspects of the problem under investigation are discussed in chapter four. Included in that chapter are the set of criteria to be used in evaluation of computer based instructional languages, an evaluation of these languages, and a list of needs of teacher-authors. Chapter five presents the rationale for and description of a new teacher oriented computer authoring language while the results and evaluation of the new language are contained in chapter six. The last chapter discusses implications for future research.

CHAPTER II

HISTORY AND REVIEW OF RELATED LITERATURE

If at the turn of this century someone had forecast that a form of individualized instruction would be provided by a distant relative of the abacus, the person likely would have been branded an idiot. But the electronic digital computer, which evolved from the abacus, has been used as an experimental instructional device for nearly a decade. Yet in this very short period of time significant changes have occurred in computer hardware and software as well as in research directed toward the use of computers as devices for instructional purposes. But how did the digital computer become involved with such an apparently unrelated field of education as individualized instruction?

The first two sections of this chapter attempt to answer the above query by tracing the histories of both individualized instruction and digital computers from their early beginnings, through the point in time when their paths first crossed in an experimental setting--this resulted in the first published article describing computer-assisted instruction. A brief review of developments in the area of programmed learning and teaching machines is presented in the third portion of this chapter while the final section views differences between teaching machine software and computer software.

I. Digital Computers

The abacus, a form of digital computer, was first used by the Chinese about 3,000 B.C. It was not until nearly five thousand years later, 1812, that an Englishman, Charles Babbage, conceived and attempted to build a mechanical calculating device. Unfortunately, the machine was never completely built and the development of computers was halted for nearly one hundred years. In 1937, H. W. Aiken of Harvard University built an electromechanical computer called the MARK 1. The first electronic digital computer, the ENIAC (Electronic Numerical Integrator and Calculator) was designed by J. P. Eckert and J. W. Mauchly at the University of Pennsylvania in 1946 (Borko, 1962, p. 42). The demand for electronic computers, primarily for scientific and data processing purposes, increased at an accelerating pace. However, it was not until 1960 that educators became aware of the latent possibilities of the computer as an instructional aid and research tool.

II. Individualized Instruction

Approximately 2,000 years ago Socrates conducted what is now generally regarded as the ultimate in individualized instruction. He individually taught his students by using a method of question and answer. This method of instruction, involving one student and one human tutor, is often referred to as the "Socratic" method of individualized instruction.

Since the beginning of the twentieth century, American education has striven for but not achieved the ideal of individualized instruction for every student. There are many reasons for not achieving this ideal of individualized instruction; the principal reason being one of logistics. Schools of Education simply cannot produce enough teachers to reduce the pupil-teacher ratio to 1-1; nor can we yet pay for this ratio.

III. Teaching Machines and Programmed Learning

In the last fifty years a number of devices were designed to help the teacher communicate various aspects of the subject matter to the student. Most of the teaching devices, ranging from the standard chalk and blackboard to recent sophisticated audio-visual equipment, were aids to the teacher but had one drawback--the devices were centered about the teacher. A few educators in the early portion of this century recognized the need for a device that could help students learn, in an individualized manner, and yet not be centered about the teacher. A major breakthrough in the search for a special type of teaching device occurred when Pressey (1926, 1927, 1932) developed a machine which could automatically teach drill material as well as administer and score tests. However, psychological theory had not yet come to grips with the learning process and Pressey's teaching machine was not widely accepted.

Pressey (1932) lamented the fact that instruments for use in education were relatively underdeveloped. He forcast the advent of new instruments and materials that would give impetus to research advances in his field. Unfortunately, as has happened to the works of many creative persons, the potential of Pressey's teaching machine had not been recognized and the machine was all but forgotten.

It was not until Skinner (1954) published an article, "The Science of Learning and Art of Teaching," that interest was rekindled in the underlying principles and potential classroom applications of programmed learning and teaching machines. In the article, Skinner described an experiment in which mechanical procedures were used to arrange intermittent reinforcement to condition the behavior of pigeons. The experiment showed that pigeons could be taught to stand on one foot, play ping-pong, dance, turn in circles, and other activities. The method of intermittent reinforcement based on principles of learning theory have since been adapted to programmed instruction. It should be noted that Pressey focused attention on the teaching machine (or hardware) rather than the "program" contained in the machine. Skinner on the other hand was more concerned with the "program" (or software) than with the machine (Stolurow & Davis, 1965).

Skinner's programmed instruction incorporated a method of individualizing instruction by allowing the learner to proceed through a lesson in a linear fashion through self pacing. The linear method requires the learner to read and respond to a frame of information. If the response is correct, immediate positive reinforcement is provided and the next problem presented. However, if a wrong response is entered, the learner is given a second attempt, a third, and so on until he responds correctly. Movement to the next problem is permitted only upon entry of a correct response. This method of progressing through a program in small steps allows for the gradual buildup of information from the simple to the more complex while allowing the learner to progress at his own rate (Skinner, 1958).

Crowder (1960) originated the technique of branching or intrinsic programming. This method was based and designed upon the concept of multiple choice responses. In this method, the learner is presented with a small amount of material and then asked to respond to a question by selecting the appropriate choice. If an incorrect choice is entered, the preceeding information is reviewed, the nature of the error explained, and the question repeated. The frame to which the learner is branched depends upon the choice selected.

Thus linear programming and intrinsic programming represented two schools of thought regarding the types of programs to use in teaching machines. In his research of literature related to programmed instruction, Schramm (1964) noted that only a handful of experiments made use of intrinsic programming as compared to linear programming. Although these two types of programs were mainly used other types were developed. Glaser (1962) developed the spiraling method of programming which introduced the concept of reviewing. Despite variations in complexity and special features most programs provided some form or variation of the tutorial or Socratic method of teaching. Basically the majority of programs: (*a*) present the individual learner with a set of information followed by a questions to be answered, problem to be solved, or exercise to be performed; (*b*) request a response from the learner thus requiring the learner to actively participate in the learning situation; (*c*) provide some form of automatic feedback as to the correctness of the response; and (*d*) allow the learner to advance at his own rate. "Typically, they proceed in small steps of graded difficulty, so that mastery of concepts, understanding, and skills are gradually built up as the student advances through the program [Lumsdaine, 1959, pp. 13-14]."

During the short span of time from 1958 to 1960 there were significant developments in the types of machines or devices used to present programs in instructional settings. Pressey's original teaching machine had evolved into four different types of teaching machines or devices: (*a*) the conventional teaching machine as originated by Pressey and advocated by Skinner; (*b*) the automated teaching machine pioneered by Briggs (1958) and Coulson and Silberman (1958); (*c*) the programmed textbook devised by Glaser, Homme, and Evans (1959) and Crowder (1959); and (*d*) the electronic computer as a teaching machine as explored by Rath, Anderson, and Brainerd (1959).

Since 1960, the educational market has been flooded by a proliferation of different types of teaching machines. Conventional and automated teaching machines have not fared too well in competition with other devices in the instructional field. This was a result of the many types of conventional and automated machines that were sold to educational institutions without the machines being thoroughly tested in experimental settings. A major shortcoming of these machines was their expensive cost, ranging in price from a few dollars to several thousands of dollars. Inherent restrictions in the capabilities of the conventional and automated teaching machines in turn restricted the types of programs that could be implemented. Also, there was some evidence that students using programmed texts performed just as well as students using conventional teaching machines (Goldstein and Gotkin, 1962). These factors brought about a decline in the use of conventional and automated teaching machines in favor of the programmed text.

After a short "second life," Pressey's teaching machine appeared to become obsolete. Because the programmed text is portable and at present more economical in terms of instructional costs when compared with computer based instruction, the programmed text will perhaps remain on the instructional scene for a short time yet. But, it would appear that in the near future the same fate is in store for the programmed text as was for the conventional and automated teaching machines. At present, computer based instruction systems are being developed to provide a means for individual instruction. Many writers well known in the field of teaching machines, such as Coulson, Glaser, Silberman, and Stolurow are now working in the area of computer based instruction.

We have seen digital computers evolve from the abacus, enter the instructional field, and offer education an opportunity to provide students with the long sought ideal--individualized instruction.

IV. Computers and Teaching Machines: Hardware and Software Differences

In the previous section mention was made of the fact that Pressey was primarily interested in the teaching machine (hardware) as it related to the instructional process while Skinner placed more emphasis upon the program (software) contained in the teaching machine. This section defines hardware and software and their related functions in teaching machines and computers.

The term hardware refers to the physical components of the computer and to any operations which the computer can execute. These executable operations are made possible by the interaction of the computer's physical components. Hardware may refer to input/output devices and memories as well as "wired" in logic. Modifications of the hardware system can only be made by modifications to physical components or by re-arranging their interdependencies through changing their physical connections. For example, in some computers addition is carried out through physical means (hardware) by having circuitry so designed as to add binary numbers. In other machines this ability to add must be specified by a human who through proper arrangement of instructions, sequences a set of internal hardware operations to produce the desired result. The specification of such instructions which sequence and organize internal hardware functions to produce the desired result is referred to as software.

In this sense teaching machines had relatively little or no software. However, if one wishes to consider instructions which are executed by the student, to modify the sequential presentation of questions by the machine, for example, then in this sense there was some software capability. In some instances a pre-coded film or paper tape which contained software was used on some of the more sophisticated teaching machines. Such pre-coded instructions were necessary for automatic branching to special instructional sequences.

Software implies, of course, a symbolism or notation which is "meaningful" to the hardware in the sense that the operation of the machine is modified. There may be many levels of software to permit an easy interface between the user and the hardware of a machine. Thus, software becomes extremely important in computer based instructional systems because it must not only permit organization of hardware for purposes of reflecting instructional strategy, but it must also permit subject matter to be presented.

CHAPTER III

COMPUTER BASED INSTRUCTIONAL SYSTEMS

This chapter is primarily concerned with the development of Computer Based Instruction (CBI) systems. Focus will center upon history of early explorations in the field of computerized instructional systems. In addition, a brief description of recent advances in computer hardware, software, and author languages is noted.

Research articles related to hardware and software components of CBI systems are reviewed by Dick (1965), Gentile (1965), Hansen (1966), Hickey and Newton (1967), and Zinn (1967b). Books edited by Borko (1962), Coulson (1962), Glaser (1965), Bushnell and Allen (1967), and Gerard (1967) provide fairly extensive coverage of issues related to research, theoretical orientation, philosophy, future prospects, and problems in the area of CBI.

The first publication describing the experimental use of a computer as an instructional device originated at International Business Machines Corporation (IBM) by Rath _et al_. (1959). This publication described a program which used a computer and an electronic typewriter to teach binary arithmetic. Since this first attempt at using a computer as a teaching machine, research in the area of CBI has accelerated.

I. Early CBI Laboratories

Initial experiments with computers as possible instructional devices were carried out prior to 1962 at the following research centers.

International Business Machines (IBM)

IBM carried out much of its early research at Watson Research Center in Yorktown Heights, New York. An IBM 650 computer, although intended for purposes of scientific calculation, was maintained at the Center and used for teaching-machine research. A number of input-output devices such as magnetic tape, punched cards, and electric typewriters were used as well as large random-access disk files for storage purposes. Twenty special typewriter terminals in the Center were directly connected to the computer. A number of terminals were located in various areas of the U.S.A. and connected by telephone lines to the Yorktown computer. The terminals were used for student-computer interaction (Uttal, 1962; Gentile, 1965). From 1959 to 1965 programs were developed in subjects such as German, Stenowriting, Statistics, Audiology, etc.

Bolt, Beranek and Newman Inc. (BBN)

Located at Cambridge Massachusetts, BBN experimenters used a PDP-1 computer (Programmed Data Processor manufactured by Digital Equipment Corporation) as a teaching machine. Input-output devices used were typewriters, a reader, 16 computer-controlled relays, an oscilloscope, and a "light pen" (Licklider, 1962, p. 218). This system was among the first to use the "erase" feature which permitted correction of typing errors. Another feature was the program mode in which the student was able to sketch a curve on a display grid with his light pen and upon completion of the sketching, the computer determined and displayed the best fitting parabola corresponding to a particular equation. BBN investigators were first to use a computer as a teaching machine to study perceptual learning (Gentile, 1965, p. 18).

Coordinated Science Laboratories (CSL)

CSL was part of an engineering department at the University of Illinois. A computer based instructional system called PLATO (Programmed Logic for Automatic Teaching Operations) was developed by Bitzer, *et al*. (1962). The system used the ILLIAC computer, a slide reservoir, the screen of a closed-circuit television system which permitted material to be displayed along with slides, and a student control panel or keyset. The keyset was similar to a typewriter and allowed the student to enter numeric or alphabetic data.

One of the main features of PLATO was its ability to instruct a number of students concurrently by using time-sharing. Other features were: (*a*) it gave students the computational devices of a powerful digital computer; (*b*) it contained a teaching logic that was determined by programs within the central computer; and (*c*) it automatically kept detailed records of each student's progress through the course material. By 1962, the PLATO system was already in its third revision.

System Development Corporation (SDC)

The experimental system at SDC, in Santa Monica, California, consisted of three major units: (*a*) a Bendix G-15 general purpose computer which read information from punched paper tape or cards; (*b*) a random access slide projector which held a maximum of six hundred 35-mm slides to display instructional materials to the student; and (c) an electric typewriter which was used as an input-output device.

Thompson Ramo Wooldridge Inc. (TRW)

TRW's Modular Information Processing Equipment System was originally developed for data processing. Using this system, investigators at TRW designed an automatic tutoring device called MENTOR. The device selected films on the basis of real-time responses, presented auditory and visual displays and scored responses automatically. Colored or black and white frames or motion picutres, drawings, text, charts, and graphs could be displayed in any order desired (Chapman and Carpenter, 1962).

An important feature of these early systems was use of terminals or response stations which served as input-output devices for student responses or information displays. This device was crucial in the early phase of these studies since the terminal represented the means of communication between man and machine. Other important features that were being used by some of the centers were time-sharing, oscilloscope, light pen, and random access projectors. These significant features were used only three years after the first attempt at computerized instruction.

In February, 1963, the Training Research Laboratory (TRL) at the University of Illinois ordered the first computer system to be used solely for experimentation in the area of computer based instruction. TRL was directed by Stolurow. The system used was called SOCRATES (System for Organizing Content to Review And Teach Educational Subjects).

It was designed to meet the requirements of the ideographic contingency model of teaching as well as for research towards testing the model (Stolurow and Davis, 1965). The ideographic contingency model assumes teaching to be divided into a pretutorial phase which selects the initial teaching program required to achieve a desired outcome and a tutorial phase which implements the teaching program and concomitantly monitors the student's performance. This model, in addition to using the student's last response, uses all other information to make decisions as to the next instructional frame or sequence of frames to be presented.

The first commercially "package" CBI system was announced by IBM in October, 1964. An author language called Coursewriter I (Maher, 1964) was designed primarily for use by nontechnical persons. Coursewriter I enabled an author, with a minimum of training in the language, to construct various sequences of teaching logic. This language provided authors with opportunities to present information, ask questions, evaluate responses in terms of correct answer, wrong answer, or unanticipated answer, and decision making power based on responses entered or numerical values of counters. The counters could be used to accumulate results of various learning outcomes. In addition, the author was able to employ a record keeping process known as a Student Record, which recorded and accumulated all student responses and response latency times.

The next two sections of this chapter present a review of the developments in the area of CBI hardware and software.

II. Current Major CBI Systems

Since 1962, all but one of the early experimental CBI centers mentioned in the previous section, Thompson Ramo Wooldridge Inc., have remained very active in the field of CBI research. In that short period of time the remaining centers have more fully utilized and further developed their CBI systems. Meanwhile, new CBI centers have evolved and others are in the process of being developed. Concurrent with the growth in the number of CBI systems, there has been a tremendous change in the hardware and software capabilities of the systems. In order to view some of the innovations and developments, this section presents a brief discussion and appraisal of hardware and software features incorporated at a few of the major CBI systems.

Hardware

In this section, recent developments in computer hardware, as related to CBI systems, are studied from two different areas. First, one must consider the nature of the central processing units (CPU) so that it is possible to discuss operations such as processing of student records, time-sharing (multiprocessing), real-time, etc. Time-sharing refers to the technique in CBI whereby a large number of people situated at terminals appear to be simultaneously served by a computer. The computer reacts so rapidly that it permits dialogue between users and machine or even among users (Fano and Corbato, 1967). Real-time is the time taken by the computer in physical processing of information to data. Processing time should be sufficiently rapid so that information is available to guide other physical processes. Some of the factors in the outcomes and benefits of equipment in CBI systems as stated by Hansen (1966) are feasibility, adaptability, flexibility, expandability, reliability, and cost.

The second aspect of hardware to be discussed in this section is the student-subject matter interface. Interface describes devices such as the two-way typewriter, cathode ray tube, slide or image projector, audio unit, etc., through which the student interacts with the subject matter (Wodtke, 1967, p. 319).

Weaver (1963) cited one of the problems in communication as the technical problem of how accurately and rapidly the symbols of communication can be transmitted from sender to receiver. The communication problem was further investigated by Glaser, Ramage and Lipson (1964). They extensively studied the relationships between interface requirements and subject matter characteristics of elementary mathematics, reading and science. In an evaluation of interface devices, Wodtke (1967) mentioned that stimulus display and response processing characteristics are two important dimensions of the interface. Since the study by Glaser _et al_. (1964), a number of authors have viewed interface devices and the related interchange time between student and computer as one of the important hardware problems worthy of further investigation and development (Dick, 1965; Hansen, 1966; Wodtke, 1967; Rogers & Gariglio, 1967).

The PLATO system (Bitzer, _et al_., 1962) under the guidance of its originator, Dr. D. L. Bitzer, has developed into one of the best known CBI systems currently in use. PLATO hardware facilities consist of an operating system connected to a CDC 1604 computer which uses magnetic tape and paper tape for auxilliary storage and has a line printer available for printed output. According to the definition of time-sharing used in this paper, PLATO, like a number of other "dedicated" CBI systems, uses time-sharing.

At present, the PLATO system has twenty student stations with video capability. The interface devices available to each station include an electronic keyset and a television screen. A student sends information to the computer by way of the keyset while computer prescribed information, intended for viewing by the student, is presented on the television screen. Information displayed on the television screen originates from two sources. The first source is an "electronic book." The electronic book projects slide images on the television screen, features a random-access slide projector containing 122 slides, and has a slide access time of 1 microsecond (one-millionth of a second). All students share the same system projector since the slides are continually scanned thus permitting students to view the same slides or different slides simultaneously (Bitzer, Lyman, and Easley, 1965).

A second source of video information, the "electronic blackboard," plots diagrams, symbols, and words on the television screen or it can superimpose the material generated over slide projections already featured on the screen. This procedure permits presentation of textual material which requires students to "fill in the blanks." Other hardware features worthy of note is the fact that the entire blackboard can be erased in two-tenths of a second and also the absence of typewriters allows student station areas to function in an atmosphere of relatively low-noise level.

Current work at the PLATO Project laboratories is directed toward developing an audio capability for the system. Also, experimental work is being conducted to develop an 12 X 12 inch plasma flat tube (Bitzer, 1968, p. 10). The new display, which exhibits memory and does not require continuous rewriting, will possibly be connected to the central computer via telephone lines (Bitzer, Hicks, Johnson & Lyman, 1967). Eventual production of the new display is expected to lower terminal costs considerably and thereby enable PLATO to expand its facilities to accomodate 4,000 remote terminals within the next few years (Entelek, 1967).

Bolt, Beranek and Newman Inc. (BBN) has put a great deal of effort into the development of software rather than hardware. The BBN System (MENTOR) uses a modified PDP-1 computer to which a number of teletype-writers are connected. Cathode ray display devices which write alpha-numeric characters and graphic displays are also used. Magnetic tape and paper tape are used for auxiliary storage.

The PLANIT System is operated by System Development Corporation. PLANIT originally used a time-sharing Q-32 computer with planned switches to an IBM 360, Model 65 and then to an IBM 360, Model 67, as equipment is acquired (Silberman, 1966). Auxiliary storage devices are drum, disk, and tape. Fifty student stations are available but only twenty to thirty can operate at peak periods of use. Most stations contain only 1 interface device--a teletype. A few of the stations include a CRT display, light pen, keyboard input and and RAND graphic tablet input.

Investigation with particular emphasis upon developing refinements in CBI interface equipment is currently being conducted at the University of Pittsburgh Learning Research and Development Center (Ragsdale, 1966). This system uses a PDP-7 computer. Research efforts are being directed toward further developments in cathode ray tubes, typewriters, random access audio units, RAND tablets, and a touch-sensitive device (Hansen, 1966, p. 591). The touch-sensitive device consists of a filmed sheet of paper on the front of the CRT--when a student puts his finger on the sheet, the position of his finger on the sheet is identified by the computer.

The IBM 1500 System (COURSEWRITER II) was the first integrated system specifically designed for educational activities. A small number of these CBI "packages" have been produced and leased to educational institutions for research in CBI. The 1500 System has as its base an IBM 1130 computer with relatively limited immediate access core storage (approximately 64,000 characters). About 5 million characters of "on-line" storage is provided by 5 interchangeable disk drives. Auxiliary storage can be provided by the addition of a tape unit. Other hardware features included in the system are a card reader punch and a line printer.

A maximum of 32 student terminals can be accomodated by the 1500 System. Each terminal can consist of a typewriter and/or any combination of the following devices: a cathode ray tube with a modified keyboard, a light pen for student responses, a film projector that shows black and white or color film images, and an audio unit that plays or records messages.

The CRT displays a maximum of 640 type characters or a limited set of simple graphic displays. All characters or graphics are displayed as configurations of white dots on a black background. This device does not allow for half-tones. Student responses to the system may be made by either using the light pen or keyboard. The film projector contains 1024 frames of 16-mm film and has a positioning speed of approximately 40 frames per second (IBM, 1967a, 1967b).

Project MAC (Man And Computer), located at Massachusetts Institute of Technology, is directing much of its research toward on-line time-sharing systems. A teletype is used as the student interface device but efforts are being made to develop a visual display terminal.

RAND Corporation is exploring the possibilities of providing increased possibilities for student response input. Under development at RAND is a vertical CRT display for computer generated materials and a horizontal tablet to be used by the student. The student communicates by writing on the tablet with a special light pen. While writing, the student watches the CRT which displays his writing. Everything on the tablet is controlled by the computer except for one spot of light which is controlled by the light pen. A special switch in the pen senses the direction of the pen and the light moves leaving a line of light thus enabling the student to construct symbols, shapes, or motion in a method which simulates writing on the CRT (Ellis and Sibley, 1967).

Software

Varied and complex software features are embodied in most of the CBI systems described in the previous section. Discussion of software characteristics implemented by these systems is an almost impossible task unless a "common base" or frame of reference is used in describing the software. In order to provide this common base, the 14 categories used in describing computer programming languages, as suggested by Zinn (1967c, p. 2), were reduced to the following categories: system operations, author operations, material logic and strategy, material sequencing and presentation, student operations, response acceptance and processing, and proctor operations.

It is noted that systems discussed in this section feature either CRT or TV type interface devices. Systems which use only a teletype interface as a communication device and/or use languages basically designed for mathematical problems will not be included in this section. However, evaluation of CBI languages, including a few mathematically oriented languages, will be included in the next chapter.

PLATO(CATO)

System Operations. The PLATO compiler, CATO (Compiler for Automatic Teaching Operation), is a FORTRAN-60 compiler which was augmented and modified for PLATO use in 1964 (Bitzer *et al.*, 1964). PLATO offers time-sharing with possibility of authors and students using the system "simultaneously." A PROGSAVE procedure enables a terminated run to be continued from its point of termination.

The method of recording study responses, called DOPE (Data Obtained for Program Evaluation), permits a permanent record of the student's number, key number depressed, mode number, and latency time. Other information can be stored and several programs are now available to analyze student response data.

Material Logic and Strategy. Since 1964, three basic types of "logic" have been programmed for CATO--tutorial, inquiry, and general (Lyman, 1968). This "logic" feature permits nontechnically trained persons, such as classroom teachers, to specify the basic logic, learning materials, and appropriate parameters for a specific lesson. Then CATO automatically compiles the lesson flow for a particular lesson. Programming all the lessons in a course becomes fairly simple because the basic logic remains the same for all lessons and only the parameters need to be changed.

The basic teaching logics offered by PLATO are:

1. Tutorial logic was primarily designed to insure that each student would pass through a fixed sequence of materials but with a flexibility for the student to branch to specified lesson elements. The student is presented with information and then asked questions about preceding material. Upon presentation of a question the student has the following main options available: (*a*) he types his answers and then asks the system to judge his answer; (*b*) he may ask the system to branch him to a help sequence; and (*c*) he may ask to be branched to a "challenge" sequence.

2. Inquiry logic presents a student with a series of problems to be solved and upon request, an option to use reference material related to the particular problem. In this approach the student is permitted to choose his own strategy in searching for information needed to solve a problem.

3. General logic represents a "combination" of the tutorial and inquiry methods with a basic feature being the many options for branching. Student control keys permit the student to continue or reverse pages containing material and problems to be solved. Other control keys allow the student to ask for help, special information, or a judgment to an answer. Provision is made for students to type a "comment" in any part of the program.

Material Sequencing and Presentation. The material presented in the main flow of the lesson logic is called the "main sequence." This main sequence displays lesson material on the TV interface by means of a series of pages and/or slides names "main pages." Some restrictions associated with main pages are that each page may contain a maximum of: (*a*) 18 lines containing 40 PLATO characters per line; (*b*) 18 typewritten lines containing 45 characters per line (slides); and (*c*) 12 questions. Also, each lesson is restricted to a maximum of: (*a*) 126 questions; (*b*) 150 pages (main, help, challenge, etc.); and (*c*) 114 slides available for use in main pages.

Four types of branching sequences are available in the logic of the GENERAL program. These sequences are:

1. A maximum of six *help* pages can be assigned to each question in the main sequence. Provision is also made for each specific wrong answer to have a special help page.

2. The *challenge* pages, up to a maximum of six, operate similarly to main pages with each page permitted answers and assigned help pages.

3. There are three basic pages used in *information retrieval*: (*a*) general reference material or data pages which may contain information relevant to problem; (*b*) dictionary pages containing definition of various terms; and (*c*) investigate pages.

4. The *comments* page allows the student to type lines of comments at any time. Usually, comments are not processed but are stored for later retrieval by the instructor.

Branch pages along with the point in the main sequence where branches may occur must be designed by the author. Students wishing to exit from any of the branch sequences may do so at any time by pressing the AHA key.

The lesson presented to the student flows along the main sequence and may branch to other sequences if provisions are made for these branches. Usually the next main page in a sequence is not presented until all the questions on the preceding main page have been "filled in" and judged to be correct.

Response Acceptance and Processing. The student communicates his responses to the system by typing his answers on a keyset which contains alphanumeric and control keys and resembles a standard typewriter keyboard. Whenever an illegal key is pushed an orange light flashes on and remains on until a legal key is pushed. Student responses are restricted to about 70 alphanumeric characters. After typing his answer the student pushes the JUDGE key which indicates to the system that the anser is complete. The system responds to the answer by displaying a two character code such as OK (correct), NC (not complete), NO (incorrect), or SP (misspelled) next to the answer(s).

Answer analysis is carried out by subroutines called judgers. Judgers enable the student's responses to be checked against the author's stored answers. The judgers are referred to by number and are programmed by the author to: (*a*) suppress spaces and judge alphanumeric responses; (*b*) test for valid fixed point integers or variable names and floating point integers or variable names; (*c*) display slides containing the correct answers; (*d*) exactly or partially judge spelling, or keywords; and (*d*) branch to special sequences or present the next page in the sequence depending upon the response condition.

It is possible for authors to allow for the "connect feature" which permits interaction between students at various terminals. This feature has been used in a number of on-line experiments (Bitzer, *et al.*, 1967). These experiments have attempted to simulate real-life experiences as would be found, for example, in political conferences.

An important software feature of PLATO is the DOPE (Data Obtained For Program Evaluation) routine. DOPE records the student number, key number, mode number (main, help etc.), time, and identification tag. The author has the option of asking for a recording of the page number, problem number, and judging result (Lyman, 1968, p. 53). A number of programs which sort and analyze student responses are available for author use.

Student Operations. Students progress through a lesson by pressing special control keys thus permitting some freedom in controlling course flow. If a student wants help with a problem he presses the HELP key and may be automatically branched to a remedial sequence, if provided for by the author. If the HELP sequence has not been programmed by the author, and the student presses the HELP key the orange light will flash on. This indicates to the student that an illegal key has been pushed. A few control keys and their functions are:

AHA returns the student to the main sequence at the point from which he branched to an auxiliary sequence

ANS provides the correct answer to a question

BKSP erases last character typed

COMNT permits student to enter comment

CONT moves student to next page in sequence or back to main sequence if last page in HELP sequence has been encountered

DATA initiates pages of useful information

DICT presents list of definitions available

ERASE erases the last answer entered

HELP begins a help sequence

INV takes the student to first page of investigation sequence

JUDGE appraises student's answer

REPL replots any blackboard writing which was previously on the TV screen

REV presents the previous page in the sequence

The student may terminate a lesson by simply leaving the station. When he returns next day, providing the lesson has been saved by the author, the student begins the session by merely typing his name and pushing the REPL key.

Author Operations. An author using the PLATO program GENERAL usually plans his lesson by completing an answer master form. The answer form contains necessary parameter information in the following columns:

USE This column designates the page as being main, help, investigate, etc.

PAGE NO. Each page must be identified by a unique number between 1 and 150.

SLIDE NO. A slide, numbered between 1 and 114 is assigned to the page on the corresponding line of the form.

EFFECTS In this column, the special effects associated with the main page are specified.

PROBLEM NO. Problems on each page are usually designated by ordinal numbers.

JUDGER NO. The number associated with a particular judger is entered in this column.

ANSWERS One or more acceptable correct answers can be listed by the author.

SPECIAL WRONG ANSWERS A maximum of six special wrong answers are permitted.

HELP PAGES This column contains the help page or help pages associated with a problem.

SPECIAL HELP PAGES The same numbers of special help pages are the numbers of special wrong answers is required.

Once the answer master form has been "filled-in," an author enters the lesson parameters pertinent to GENERAL logic on-line from a PLATO terminal. Due to the length of program GENERAL, authoring procedures are usually carried out in three steps.

Normally, the first step includes use of a program, called GENAUTH, which provides the author with a great deal of flexibility in programming lesson parameters. An author is permitted to: (a) enter, advance, delete, or insert main page numbers; (b) indicate special effects and challenge sequences associated with a particular main page; (c) specify a list of eight character dictionary terms available to the student; and (d) enter lists of names to be displayed in the investigate sequence.

In the second step the author describes each unique lesson (global) page by assigning slide numbers and problems to the global page. A problem description consists of judger numbers, answers, special wrong answers, help page numbers, and special help page numbers. Help pages are described in a manner similar to the description of global pages although help pages cannot have help sequences associated with them. Also, dictionary and investigate pages are assigned page numbers and corresponding slide numbers.

Finally, the third authoring step uses a program called CONSTNT. In this step the author specifies special arrays of words to be used as acceptable keywords in the investigate subroutines.

An author entering lesson parameters may use these three steps in any order. The parameters are stored on magnetic or paper tape. Since input parameters are very detailed, the author may check lesson parameters by using a computer program called PRINT which uses the computer line-printer to provide a listing of the parameters. The author wishing to make parameter deletions, insertions, or replacements may do so by using the appropriate steps to modify the parameter tape.

BBN Mentor System (MENTOR)

The MENTOR system (Feurzeig, 1964, 1965, 1967; Swets, *et al.*, 1965), developed by Bolt, Beranek, and Newman (BBN), uses a language called MENTOR which was primarily designed to permit teacher-authors to program conversational tutorial dialogues. MENTOR is embedded in the list processing language LISP and offers authors a powerful and convenient way to program complex textual material. Essentially MENTOR is a verbal language rather than a computational language. Another instructional language currently used by this system is a flexible on-line computational language called TELCOMP (Myer, 1967).

System Operations. MENTOR uses a modified DEC PDP-1 computer, LISP compiler, and MENTOR interpreter. Although MENTOR is used with the PDP-1, it can be used with smaller and slower computers. Terminals consist of a teletype keyboard and a display screen.

Material Logic and Strategy. The author wishing to construct an instructional program must first conceive of a model of the problem and define pedagogic strategies to be used in the interaction between student and system. Next the author must specify the possible contexts and possible inputs within that context. Inputs allowed for the student are listed in the form of acceptable questions and declarations. These lists may be as extensive as desired by the author.

In a program designed to study problem-solving, the emphasis is upon having the student gradually acquire information necessary to solve a problem by having him make inquiries or declarations in response to questions put to him by the computer. The author may devise conditional strategies so that the system answers "good" questions, reproves hasty conclusions, questions grounds of inference, suggests new approaches, etc. In this way the student is able to converse with MENTOR while attempting to solve a problem posed by the system. Thus the student may view MENTOR as both a guide and a critic.

Student Operations. To begin a session, the student types his name and the course he wishes to work on. MENTOR checks the student's performance record and poses an appropriate problem or question. The type of student questions or assertions that may be made in response to MENTOR's queries are limited to a specified vocabulary permitted by the author. This means that student responses must be made according to the format of acceptable input statements. After processing an input statement, MENTOR examines the student's previous actions as well as information he possesses and then responds in natural English by posing a question or by making an assertion or comment.

MENTOR continually evaluates a student's performance during the session. Thus, when a student signs off, the system automatically records the student's evaluative data and a brief list of areas requiring further study in a particular topic (Feurzeig, 1967, p. 84).

Author Operations. Instructional programs are prepared by using an on-line teletype. Authors are usually encouraged to construct their programs in a flexible manner so that a student progressing through a course feels that MENTOR is under his control. If desired the author may specify nonverbal material such as tables, graphs, pictures, and audio messages.

PLANIT

PLANIT (Programming LANguage for Interactive Teaching) is a user-oriented author language developed at SDC (Feingold, 1966, 1968; Feingold and Frye, 1966; Silberman & Rosenbaum, 1966). Originally, PLANIT was designed to enable teacher-authors to develop instructional lessons in statistics. However, the language was so flexible and powerful that it has been used by authors in developing courses in history, spelling, psychology, etc. (Feingold, 1968, p. 44).

System Operations. The PLANIT system currently operates on the SDC Q-32 time-sharing computer with plans to change to the IBM 360/67 System. Feingold (1968) indicated that SDC plans to develop PLANIT to operate on any IBM 360 system which is equivalent to or larger than the IBM 360/40. This new version of PLANIT would accomodate 50 to 100 students simultaneously and would operate as one of the following types of systems: (*a*) a dedicated system; (*b*) in connection with an operating system; or (*c*) under the control of a time sharing system.

PLANIT is written in JOVIAL and the system uses the JOVIAL compiler. Programs and student records can be stored on disk or tape. Student records are updated and stored on disk only if the student indicates to the system that he has "finished" a particular segment of a lesson. Lost material or records cannot be recovered. The author can copy student records from disk to tape but student performance records cannot be saved directly onto tape. When a student completes the sign-on, PLANIT automatically continues from the frame the student has "finished" previously and keeps track of the student's progress throughout the lesson.

The terminal interface device used is usually a teletype although a graphic input tablet has been modified so that a CRT image can be projected on its surface. However, these tablets have been found to be too fragile and expensive for general use (Frye, 1968, p. 37).

Material Sequencing and Presentation. A lesson author would use either the command, lesson building (CO), calculate (CALC), or execution (EX) operation modes. The PLANIT command and lesson building modes permit an author to construct, edit, and store a lesson in the desired sequence while the EX mode is required to present the lesson to the student. Finally, the CALC mode can be used either by the author in calculating a numerical value or by the student in executing the lesson.

One feature of PLANIT is that it can provide the course author lesson building information while the lesson is being constructed. Information, if requested by an author, is provided at two levels. The VERBOSE level provides fairly elaborate information regarding construction rules and is usually used by beginning authors. Experienced programmers normally work at the CONCISE level which supplies the author with a minimum of coded information.

In PLANIT, a lesson is composed of a series of frames. Each frame is composed of groups and each group is composed of lines of information which can contain text, questions, answers, specifications, etc. There are five basic types of frames that can be presented in a lesson (an example of the concise level appears in parenthesis): (P)roblem, (Q)uestion, (M)ultiple choice, (D)ecision, and (C)opy.

The P frame was originally designed for specialized use in teaching statistics but now can be used for presenting three kinds of information.

Authors can specify population parameters for generation of random samples data related to normal or skewed distributions. The student can then be asked to compute various statistics such as range, mean, variance, etc. A second use of the P frame is that a set of "hints" called "STEPS" may be made available for students having difficulty in solving a particular problem. Thus a student can type the word STEP and may receive a hint to help him correctly answer the problem. Finally the author can provide the student with control over mathematical functions required to solve a specific problem (Feingold, 1968, p. 47).

The Q and M frames are constructed in a similar manner except the M frame lists a series of possible answers and the student is required to select the most appropriate choice. In the Q frame a question is asked and the student must supply an answer. The logic of these two types of frame is similar to the answer format of COURSEWRITER. The author can enter anticipated correct, wrong, and unrecognizable answers and specify the appropriate action corresponding to each of the answers. Four types of action commands are available to authors constructing a lesson:

F--This action permits an author to feedback an appropriate message to the student regarding the correctness of the answer. If the author does not include any text following "F," PLANIT will randomly select an appropriate message from two lists, one for correct answer and the second for incorrect answers.

R--This letter provides a repeat action in that a message will be generated to the student and the system waits for the student to enter another answer. The author has the option of specifying the message or of defaulting the text and using the system comment "WRONG, TRY AGAIN."

C--PLANIT prints the comment: "THE CORRECT ANSWER IS----."

B--This instruction initiates the branch operation. Since all frames are numbered or labelled, a branch may be made to any frame number or label. PLANIT also provides the author with the facility to branch a student to another lesson. Upon encountering the end of the lesson or the command "B" followed by no specified location for branching, PLANIT will return to the next frame in original lesson from which the branch was made.

Branching decisions are made available to the author by the D frame. PLANIT permits an author to check a student's performance regarding response latency, errors, functions used, path taken through the lesson, etc. over any portion of the lesson and to specify branching conditions based on this performance data (Feingold, 1968, p. 44). The C frame aids the author by allowing him to build a frame which is a copy of a modification of a frame already used in the lesson.

Response Acceptance and Processing. PLANIT accepts alphanumeric and special characters as input from the electric teletypewriter. During construction of each frame the author indicates to the system whether the student's response is to be a literal string or whether it is to be an expression or function which must first be evaluated. The author must prefix all lines in an answer set by either a letter or a numeral from 1 through 9. An answer set prefixed by letters indicates to PLANIT that the student's response will be in literal form. If the answer set is tagged by numerals, then PLANIT assumes the author's entry is an expression or function and must first be evaluated before the student's response is compared with it.

Correct answers are indicated by a plus sign (+) inserted between the answer and the answer line identifier. Therefore, if the student's response matches the prescribed correct answer for the frame, PLANIT will search for the corresponding line identifier in the action group and carry out the operations designated by that line. Provision is made for the author to specify appropriate action for an unrecognizable or unanticipated response. This is achieved by omitting the line tag and prefacing the action statement by a negative sign. If an unrecognizable answer is entered, the system usually generates an appropriate random message from a specified list and repeats the problem or types the correct answer.

Five PLANIT commands permit the author with the facility to specify a wide range of answers rather than restricting answers to a particular set of anticipated responses. The WAIT command permits a lesson author to specify maximum latency response time in seconds or minutes. If the student has not responded before expiry of the WAIT interval specified, PLANIT will advance to the action group and search for the line indicating time out procedures. A non-tagged statement in the action group, prefaced by an apostrophe, is used as to specify an action such as give the student a "hint" or branch him to another frame. A second command allows a numeric response to be judged as an answer match if it lies "WITHIN" a particular numeric interval of the prescribed answer.

The remaining three commands or services routines are PHONETIC, KEYWORD, and FORMULAS. When these routines are turned "ON," the following occurs: The PHONETIC routine reduces both the author's prescribed answers and the student's response to encoded answers and then compares the encoded messages for a phonetic match; the KEYWORD routine searches the student's response for words that match the prescribed answer in a specified order; and the FORMULAS routine permits PLANIT to check for an algebraic match between the student's response and the prescribed answers. Any combination of these operations may be used in response analysis. For example, if the author wished to check for keywords but was not concerned with their order, then he would turn on both KEYWORD and FORMULAS routines.

Student Operations. The student who interacts with PLANIT may do so by alternating between the EX (execution) and CALC modes. If the student is required to perform arithmetic computations to arrive at an answer he may do so by typing a left pointing arrow (←) followed by an arithmetic expression or an appropriately defined mathematical function. When the student presses the return key, indicating that he has completed his answer, the machine performs the specified operation and types the calculated answer almost immediately. To enter his answer the student must return to EX mode by typing an "up arrow" (↑) followed by the answer which may be a numeric value, mathematical expression, literal string, or letter corresponding to the appropriate choice in a multiple choice frame.

If the student is not sure of the mode in which he is operating he needs only to type a question mark (?) when a response is required and PLANIT will respond by telling him what is expected. While interacting with a problem frame the student is permitted by PLANIT to enter the CALC mode to ask questions of a general nature. The student's question is communicated by PLANIT to a program called "PLQST" which tries to match meaningful words from the question with a previously stored base of information and then generate a meaningful response. By striking the "?" key the student may be provided with further relevant answers to his question.

Another feature of the P frame is that it provides the student with an option to ask for hints to help him in solving the problem. This is achieved by entering the CALC mode and typing STEPS. The machine replies by typing a hint to the student from a list entered by the lesson designer.

One of the most important student operations occurs when the student wishes to leave the lesson at the end of the sitting. Usually he is required to enter the CALC mode and type FINISHED. This operation automatically updates the student performance record on disk. If the student does not perform the FINISHED operation, his record for that particular sitting will be lost.

Author Operations. The lesson designer prepares a lesson on-line from a teletypewriter and is permitted to alternate between command, lesson building, calculate, and execute modes. If the designer is not sure of his present mode he may type "?" and PLANIT will display a diagnostic message describing what is expected. As mentioned previously, a lesson is composed of frames, groups, and lines. The author constructs a lesson using five types of frames: problem, question, multiple choice, decision, and copy. After selecting the type of frame desired, the author is led through a specified series of steps by PLANIT which interacts with the author and provides him will various diagnostic messages. These diagnostic messages may be quite detailed if the author is in the VERBOSE mode or abbreviated if in the CONCISE mode.

While in the command mode the lesson designer may alter a frame or group of frames by using the EDIT commands: print, delete, or insert. Also, this mode permits an author to save or delete a lesson on disk or tape. The author can not directly save a student's record since this record can only be saved by a command from the student himself. If the student has saved his record for the sitting the author can then transfer the record from disk to tape.

IBM 1500 System (COURSEWRITER II)

System Operations. The operating system (supervisory program) of the IBM 1500 Instruction System functions under the directions of a programming language called COURSEWRITER II (CW). Some of the functions carried out by the supervisory program include: (*a*) storing and maintaining program data; (*b*) providing each station with the facilities of the system; (*c*) notifying the proctor of a student's request for assistance; and (*d*) accumulating recordings of the student's performance.

Two major programs used by the 1500 System are the CW assembler and CW interpreter. The assembler translates and assembles CW statements into a form that can be executed by the interpreter. COURSEWRITER statements assembled during processing are automatically stored on disk. Instructional courses may be assembled on-line from either a CRT keyboard or a typewriter similar to an IBM SELECTRIC. Another method of assembling a CW instructional program is to enter CW statements that have been punched into cards.

One of the features of this system is the program which automatically maintains a record of disk of the student's performance during an instructional session. Information to be recorded is usually specified by the author when he registers a student for a course. A listing of the performance recordings for students taking the course may be printed on the 1132 printer. It is possible for a programmer with knowledge of FORTRAN and CW ASSEMBLER to rewrite portions of the listing program and thus enable the format of the record listings to be altered into a form desired by the teacher-author.

Material Sequencing and Presentation. COURSEWRITER material is designed to be organized similarly to that of classroom instruction. Usually course material consists of a maximum of 128 "chapters" (or segments) which are comprised of small independent units called "lessons." Each lesson contain "problems" which present concepts and associated questions. These problems enable the author to break difficult concepts into less complex problems and then based on the student's performance to vary the instructional pace to the rate of each individual student by skipping problems, branching to a new lesson or a remedial lesson, repeating a lesson, etc. Although developed as if one student were taking the course, various alternative paths must be provided for the many combinations of progressing through the course.

COURSEWRITER permits use of either the typewriter or a combination of CRT and light pen or keyboard, image projector, and audio unit. This language does not contain any "built-in" teaching logic as does PLATO. Thus the author must first organize the course into its basic teaching units. Then he must decide on the type or types of teaching logic he wishes to use, interface devices to be used at specific points in the presentation, and which kind of input device to use--keyboard, lightpen, or audio.

Before the course is programmed the author usually codes, onto a special coding sheet resembling a picture of the CRT, all the material that may eventually appear on the CRT during a course session. Then he begins coding the instructional and content portion of the course which must be in a format conforming to CW's control facilities. Once a portion or all of the program is coded, the author enters the CW instructions into the system. Each instruction contains from one to four major parts depending upon the operation it is to perform. The four major parts of a CW instruction are the: (a) label--used to identify specific instructions; (b) operation code--a two letter mnemonic describing the principal computer operation; (c) modifier--a single letter identifier which permits an operation code to perform several functions; and (d) parameter field--which may consist of a group of parameters which supply additional information regarding the required operation.

Usually, program statements are executed sequentially by the CW interpreter but an author may specify a procedure whereby the interpreter "branches" within the chapter to a statement not in sequential order. This instruction is called a branch (br) and may occur as a result of a student's response or his performance in a particular portion of a course chapter. Three types of branching statements are permissible in CW: (a) explicit or unconditional branching--this is a statement initiated by the author which directs the interpreter to a specific location in the chapter; (b) conditional branching--occurs only if a specified set of conditions are met; and (c) implicit branching--is initiated by the Course Flow Decision Table (IBM, 1967a, p. 37). The first two kinds of branches may be made to one of four reference points:

(a) a label;

(b) the point in the program where the student last entered a response;

(c) a return register; and

(d) the Nth next problem or the next problem designated as a restart point.

An author may wish to branch a student to a specific lesson in the chapter (i.e., review lesson) which is entered from several different portions of the chapter. If he then wishes to return the student to the point from which he had originally branched, the author may do so by using any one of six return registers. These return registers provide an author with the facility to vary his course and perhaps allow the student to ask for help, review, answer, etc., and to return to the point of entry if need be.

Since branching may occur only within a course chapter (segment), a "transfer" command is used to permit movement between chapters. Thus, a student may be transferred (tr) to the next sequential chapter, a remedial chapter, or a chapter containing a specialized routine but without the provisions of being returned to the original chapter by using a previously loaded return register.

COURSEWRITER can be used to present various dictionary characters and graphic displays on the CRT. The standard keyboard includes upper and lower case alphabetic characters, numerical characters, and special characters. Thus an author wishing to present a foreign language course may do so by coding the dictionary characters required by the language and associating (equating) this dictionary to the course.

Occasionally an author coding his program discovers that groups of common statements are repeated a number of times with little or no modification. This type of repetitive coding can be prevented by using CW's "macro" facility which enables an author to group sets of commonly used instructions and their variable parameters into macro routines and store them on disk. Whenever a particular group of instructions are required during program entry, the author would enter the macro name and required parameters thus causing the CW assembler to assemble the macro statements with the given parameters as part of the program. Proper use of the macro facility can reduce the time required by an author to code and enter certain repetitive routines.

Response Acceptance and Processing. Instructional programs which use COURSEWRITER often consist of textual material followed by requests for the student's response to the material. The student's response may be accepted either by the audio unit, typewriter, instructional keyboard, or light pen. Only responses entered via the last three devices can be processed by the system.

If a response is to be entered from the light pen, a "P" is displayed in the lower right hand corner of the screen. Input from the instructional keyboard is accepted whenever a "K" is displayed in the lower right hand corner and a cursor appears in the position on the CRT where the first character is to be entered. As each character is typed, the cursor moves to the right and is replaced by the typed character. A request for an enter and process from keyboard may be modified so that the cursor can be inserted onto a displayed line and the student "fills-in" the answer.

Once the student's response has been entered, it may be processed and analyzed by using the implicit branching procedures implemented by CW's Course Flow Decision Table. An important constituent in this answer analysis procedure is the answer set stored by the author, which includes, in the form of separate entries, all the anticipated correct, wrong, additional, and unrecognizable responses. A keyboard answer set contains keyboard characters while a light pen answer set includes CRT coordinates.

All instructions in the answer set are called majors. Each major is usually followed by a group of minor instructions. COURSEWRITER answer analysis executes any major which matches the student's response as well as all minor instructions until the next major instruction.

Special functions, composed of instructions coded in 1130 assembler language, are available to help perform special processing not provided for by COURSEWRITER. If a function is called by a statement in the program this initiates the Course Flow Decision Table for Functions (IBM, 1967b, p. 48), executes the function, and then returns control back to COURSEWRITER. The following is a list of a few routines offered by some of the functions and their effects on a student's response:

--upshift or downshift all alphabetic characters

--replace any group of synonyms with a fixed word or group of characters

--delete or replace punctuation or specified characters

--extract an integer portion and place in counter

--move the response into a specially designated buffer

--determine if a numeric answer is within specified limits

--keyletter match or percentage match

--key word match ordered or unordered

Other functions allow an author to generate random numbers within specific limits and place them in counters, move the contents of a counter into a buffer so that the number may be displayed on the CRT, connect two or more terminals to permit terminal interaction, etc.

Student Operation. The student must first be registered by a proctor and assigned a special student number for the course. To sign "on" a student types the course name and his student number. When notified by a message, on the CRT, that he has been signed-on the student proceeds with the course until he wishes to leave. To terminate the session he signs "off" and leaves. If the student signs himself on to the course he will begin from the last restart point encountered in a previous setting.

Instructions to students regarding procedures for terminal operation vary with the ability level of the student. Usually students are instructed: (*a*) to recognize the particular type of input device the system is ready to accept in response to a request for an answer; (*b*) how to erase portions of their keyboard responses, and (*c*) how to "enter" their responses.

Author Operations. COURSEWRITER programs may be entered on-line from a terminal keyboard or off-line by punched cards. When an author "signs on" he is automatically provided with a set of author commands in one of two control modes--assembly and checkout. In assembly mode, the CW assembler permits an author to "*enter*," "insert," "replace," "delete," "move," or "display" statements in the program. The checkout author mode enables the author to assume a student's role and checkout the flow and presentation of the program. This mode allows an author to: (*a*) "execute" any part or parts of the program; (*b*) "load" switches, counters, buffers and return registers with pertinent data; and (*c*) "show" the contents of switches, counters, buffers, and return registers. When an author signs off, the course program is automatically updated and stored on disk.

Proctor Operations. A proctor is usually the person in charge of the system during the periods of operation. While proctering he is the priority user meaning that he has control over any author or student station. This person often has a general knowledge of the operating and instructional system as well as the course material being conducted at a particular setting.

The proctor is responsible for registering course segments, authors, and students for specific courses. He assigns performance recordings for students registered in designated courses, controls the maximum course session time for each student and may provide assistance by communicating on-line with students through the instructional stations.

CHAPTER IV

THE PROBLEM

This chapter deals mainly with an evaluation of CBI languages from the viewpoint of the teacher-author. A major drawback encountered while attempting to evaluate CBI languages was the lack of a standard set of criteria. Thus, the first section of this chapter contains a set of criteria, formulated by the writer, as well as a number of guide lines to be used in evaluating CBI languages. Section two includes an evaluation of a number of CBI languages and lists a few features from these languages that would possibly be desirable in a CBI language. A discussion of current needs of CBI authors is presented in the final section.

I. Criteria for Evaluation of CBI Languages

A common set of criteria has been devised in order to facilitate a more objective evaluation of CBI languages. Since this thesis deals primarily with the educational uses of CBI in developing environments for learning, the criteria perhaps show a slight bias toward the teacher-author point of view. The criteria used in this evaluation include: (a) meaningfulness, (b) extent of separation of logic and data, (c) ease of handling subject matter, (d) ease and power of response analysis, (e) maintenance of and access to student accounting information, and (f) control over various types of interface devices.

Meaningfulness

A prime requisite in any CBI language is that it be meaningful to the needs and activities of the principal users. If CBI is to be used in schools then it is imperative that teachers not spend a great deal of time in learning to use a language or in becoming "expert" programmers. A language, if it is to be meaningful, should be natural, flexible, and oriented towards the main activities of the principal users. A teacher should be able to use meaningful words rather than abbreviations or code words to specify operations to be executed in a CBI program. Instead of using terms that are foreign to the ongoing activities of teachers, an ideal situation might be one in which CBI languages provide terms that are familiar to and commonly used by teachers. However, this means that, as in any other computer language, the words need to be precise in their meaning.

Another aspect of a meaningful language is the provision of succinct error messages to pinpoint specific errors and specify remedial courses of action. In addition, if more than one type of error occurs in any input statement then unique error messages should be provided with reference to that statement.

Extent of Separation of Logic and Data

It seems odd that programmers who use languages such as COBOL, FORTRAN, PL/1, etc., design programs that can be used again and again with different data sets. Once a statistical program has been written to calculate correlation coefficients between n-variables, then that program can be used as many times as required with different sets of data. On the other hand, most CBI programs, with the exception of those using PLATO, tend to combine their logic and data into single programs. This means that a teacher-author who writes a CBI program and then wishes to use the same basic logic with a new data set must rewrite the entire program. However, teacher-authors simply can not afford the luxury of spending a great deal of time in writing redundant code. Therefore, the logic and data (subject matter) should be separate portions of a program.

From a CBI point of view, it would be ideal to make available various logic programs that have been designed, written, tested, and researched by teams of educational psychologists, learning theorists, curriculum specialists, teachers and experienced programmers. The teacher-author would then select a logic he desires and provide the data (subject matter).

Separation of logic and data could provide educational institutions with a powerful research tool to test various teaching strategies and provide modifications to learning environments in an attempt to furnish instruction suited to the needs of individual students. Thus the team producing the logic would be given the sometimes difficult and time consuming task of producing complex conditional branching based on the student's past history and current status in the course. The teacher-author, on the other hand, would only need to match subject matter to the logic requirements.

Allowing teachers to furnish subject matter insures, somewhat, that teachers will be involved in the learning process. Also, the teacher's personality would possibly permeate each program through his method of presenting subject matter, asking questions, and providing reinforcement messages.

Ease of Handling Subject Matter

Most teachers who use a chalkboard as a means of presenting written material would likely be dismayed if they were first required to write their daily lesson notes on a coding sheet representing the classroom chalkboard. Imagine the added consternation if they were next asked to write their pre-coded notes on the chalkboard so that each line of text would be an exact duplicate of the lines of text on the coding sheet. The final blow, of course, would occur when a number of teachers find that a portion of the chalkboard contains material that may not be erased. This means that the textual material would need to be recoded before it could be written on the chalkboard.

The above example may sound far-fetched, yet this procedure is often carried out by teacher-authors who wish to present textual material on a cathode ray display screen. Thus it would perhaps be safe to say that the ease with which the language handles subject matter is of great importance to all teacher-authors.

Ideally, a language should permit an author to enter textual material in large blocks varying in length from a single word to a complete chapter. The language compiler would then edit teacher supplied textual material and format this material so that it conforms to the screen area specified by display parameters in the logic. Thus if there is need to change the portion of the screen to be used for display purposes then only the logic coordinates would need to be altered.

Further, the language should be designed so that the author himself decides to what depth he wishes to learn the language. It should be possible for a teacher to learn only basic fundamentals regarding data entry. This means that he would provide data to match a very simple pre-programmed logic which does not use complex branching and relies mainly on logic default options. Moreover, the language should be easy enough to learn so that those who wish to use more sophisticated logics may do so without having data entry become much more difficult. It should also be possible for the teacher to begin writing his own logic or to modify available logics in order that they meet his particular needs.

An author should easily be able to debug or edit subject matter that has already been processed. When the author executes his program and decides to add, delete, replace, or modify any portion of the subject matter he should be able to do so either on-line or off-line. In addition, the language should provide numeric capabilities and the facility to insert program comments for documentation purposes. Comments should appear only in the hard copy course listing. Also, the author should easily be able to change spacing between lines of text, to indicate hyphenation or non-hyphenation of words, and to specify the beginning of a new screen of text. Therefore, the language should be extremely viable in that the subject matter and/or the logic could be easily revised or modified.

Ease and Power of Answer Analysis

Included in the evaluation of a language with regard to response analysis are: the ease with which responses may be entered by an author, available answer matching routines, the power and sophistication of these routines, and the associated logical branching that may be specified. These features are very important to CBI since meaningful student-teacher interaction is made possible by the routines associated with response analysis.

The logic should specify the parameters to be used, conditional courses of action to be followed, and perhaps the accumulation of various student performance data. For example, the logic might specify the following action if a correct answer has been encountered:

```
IF CORRECT THEN PERFORM
    DISPLAY MESSAGE CORR251
    PAUSE 10 SECONDS
    PLAY MESSAGE TAPE 251
    IF CORRECTS IN LESSON>12 THEN GO TO LESSON 2
END PERFORM
```

A teacher using the above portion of logic would necessarily supply only the correct answer(s) and the accompanying reinforcement message to be displayed on the keyboard or graphic screen.

Also, the language should permit a logic programmer to specify, for example, that all keyboard responses first be edited in a designated manner before any analysis is conducted. This editing feature would then be considered as being enabled for the entire program unless it is overridden for small segments of logic. Further, the logic programmer should be able to indicate that during analysis of keyboard responses only responses containing at least a given percentage of correct spelling be considered acceptable; or if a group of words is entered that they must be in the same order as indicated by the teacher-author supplied answer.

It should be possible for the teacher-author to have the option either to provide his own reinforcement message which reflects his personality as well as suits the student response; to default to an appropriate system message randomly selected from a pool of items; or to provide no message at all.

Languages which permit light pen responses add further complications to the ease with which an author may specify a portion of the screen to be associated with a particular type of answer. Usually coordinates corresponding to an answer must be specified in the code rather than the answer(s) as is the case with keyboard responses. Beginning teacher-authors should not be required to enter screen coordinates to represent an answer. Perhaps a meaningful labelling scheme should be allowed for indicating light pen responses.

Finally, the language should be flexible enough that a logic author, in certain circumstances, could specify a hierarchy of response analysis be conducted and that complex conditional branching be carried out on the final outcome of that analysis.

Maintenance of and Access to Student Accounting Information

If CBI is to provide the means for individualizing instruction then a file containing pertinent facts regarding each student's progress through the program must be maintained. To be of value in deciding courses of action, the student's performance file should be accessible during any portion of the program. This means, therefore, that the language should provide some easy method of automatic maintenance of a performance file without the author having to repeatedly specify that a particular numeric value be added to a counter or that a switch be turned off. In addition, it should be possible to have auxilliary service routines provide the author with a hard copy listing of the group and each individual student's progress through the course, responses made at particular entry points, particular answers from the teacher supplied answer set that were matched, and the status of various counters, switches, buffers, etc. Finally, the language should provide the teacher with a method of insuring that when the students return the next day, they continue from a logical starting point within the lesson they left the previous session.

Control Over Various Types of Interface Devices

As the number of communication devices controlled by a CBI language increase then, hopefully, the more versatile the language becomes in the types of presentations that could be made and in the types of persons that could be reached. However, from an author point of view, the greater the number of devices controlled by a language then, generally speaking, the more difficult it is to learn how to use the language. Yet, this should not be any more difficult than that required of a teacher who wishes to use various multi-media devices in the classroom.

One major aspect related to the use of the interface devices is that the author should be able to maintain control over the devices. For example, if the system harware characteristics are such that film or tape search is slow then it should be possible for the author to control each device during the session so that the tape or film is always positioned prior to its actual use.

II. Evaluation of Current CBI Languages

The evaluation of CBI languages has been restricted primarily to the four languages discussed in Chapter III, due to the following factors:

1. Author languages used by computer based instructional systems have grown rapidly in number during the past decade. Zinn (1968, p. 23) stated that nearly 30 computer languages and dialects have been specifically developed for use in the area of computerized instructional conversation. Documentation of a number of these languages, including some of the "established" languages, however, is quite sparse and at times difficult to obtain.

2. Languages such as APL (Iverson, 1962; Hunka, 1967; Falkoff and Iverson, 1968), BASIC (GE, 1967a, 1967b), and TELCOMP (BBN, 1966) are basically mathematical and as a rule do not provide simplified system functions for answer matching, performance recordings, restart points, etc., which are available in some of the conversationally based languages.

An important evaluative point in discussing CBI author-languages concerns the availability of the system used as a base for each language. Currently COURSEWRITER II, PLANIT, APL, BASIC, and TELCOMP are being used by a number of establishments, but of these languages only COURSEWRITER II (the IBM 1500 System) makes extensive use of the CRT as an interface device. IBM produced the present version of the 1500 System for purposes of research and development and has leased these "packages" to thirty higher educational institutions in Canada and the United States. A maximum of thirty-two student terminals are available with each 1500 System.

PLANIT presently runs on SDC's Q-32 time-sharing system and has been or is being used via remote terminals by eight establishments in various parts of the United States. SDC plans to make PLANIT available for use on the IBM 360/65 and recently this corporation was awarded a contract, by the National Science Foundation, to implement PLANIT on the IBM 360/40 or on equivalent or larger machines. This new system could possibly handle 50 to 100 student terminals and will be able to act either as a dedicated system, in connection with an operating system, or as part of a time-sharing system (Feingold, 1968, p. 47). No mention has been made as to whether this new system will include student terminals containing CRT and light pen devices, but SDC researchers have indicated that an experimental graphic input tablet, modified to project CRT images, proved too fragile for general use (Frye, 1968, p. 37).

PLATO and MENTOR are each used at only one location--the Computer-based Educational Research Laboratory at the University of Illinois, and Bolt, Beranek, and Newman Inc., at Cambridge, Massachusetts. The PLATO system uses a CDC 1604 computer as the central control unit along with special hardware features which permit projection of film images and computer generated displays onto a CRT device. BBN's MENTOR uses a modified DEC PDP-1 computer and occasionally employs a CRT display device. Author-languages such as PLANIT, APL, BASIC and TELCOMP are currently available, on a rental basis, to remote users. PLANIT (SDC), BASIC (GE), and TELCOMP (BBN) remote users are connected by teletype terminals, to the various systems offering these CBI languages.

Two versions of APL are presently available at a few establishments using the IBM System/360 or the IBM 1500 Instructional System. APL/360 is used with the IBM 360 computer, Model 40 or larger, and is occasionally available on a rental basis to remote users via electric typewriters connected to a central computer through either a "dial-up" telephone network, leased telephone line, or by private line. APL/1500 is not available to remote users but may be used via any terminal in the 1500 System's terminal configuration.

Meaningfulness

A primary concern of teachers wishing to author CBI course material often focuses upon the problem of time required to learn the language and the ease with which the language may be coded. (Frye, 1968, p. 35) stated that an author could learn CW II or PLANIT ". . . well enough in two or three hours to begin writing his instructional lessons easily." This statement is quite misleading, at least as far as learning CW II is concerned, since most authors require that length of time to learn how to display a line of text on the CRT. Each CBI language, perhaps is not difficult to learn, but they all involve a large amount of detail that must be learned if an effective instructional program is to be produced.

PLANIT actually guides an author by reminding him of choices available to him as he enters his program. However, an author must have some understanding of frame types, author modes, answer analysis action specification, branching procedures, function specifications, matrix declarations, etc., before he can begin writing lessons other than those similar to linear type drill and practice. Authors learning CW II usually spend much of their time alternating between the Course Planning Guides (IBM, 1967a, 1967b) and the on-line instructional keyboards. To be able to write an effective CW II lesson a beginning author must learn concepts related to author control commands, format of instructional statements, problem presentation, response requests, macro building, graphic and dictionary displays, functions, timing control, etc. Beginning PLATO authors initially learn that PLATO lessons are divided into main, branch, and help sequences.

From a beginning author's point of view, numerically oriented languages use meaningful operators and therefore are perhaps slightly easier to learn than the languages discussed in Chapter II. It should be remembered, however, that languages such as APL, BASIC, and TELCOMP are designed more as student computational languages rather than as a base for presenting various types of instructional course material. Of these three numeric languages, APL is possibly the easiest to learn due to the fact that the symbolism specified is that customarily used in mathematics and logic. Therefore, students can use the language for simple numeric computation or to define functions similar to those used in other programming languages.

Most CBI languages do not provide meaningful error messages. Only PLATO and TELCOMP offer explanatory messages while others merely provide brief error codes.

Extent of Separation of Logic and Data

Once an author-instructor has learned the language and is ready to begin planning, organizing, and programming a lesson, he runs into the problem of assembling lesson content according to some type of instructional strategy or logic. Different authors often desire different convenience and capability factors when specifying instructional statements. PLATO authors, for example, select one of three instructional logics and then merely provide the text, slides, answer processing rules, and appropriate parameters. On the other hand, authors using PLANIT must plan the sequencing of frame types, but are led through a series of steps requiring the author to specify text, problems, answers, questions, branching decisions, etc. The MENTOR author is required to first conceive of a model of the problem and then to define pedagogic strategies to be used in student-computer interaction.

COURSEWRITER II requires the course author to define his own program logic in a manner somewhat similar to PLANIT and MENTOR. The inability of being able to separate logic from content often provides beginning CW II authors with a great deal of difficulty in programming instructional material. In order to be able to work concomitantly with logic and content, authors are often encouraged to "flow chart" the lesson and take into account factors such as lesson objectives, course material, media to be used at specific points in the program, scorekeeping, branching locations and conditions, etc. Therefore, in contrast, an author wishing to plan a CW II program must devote a great deal of time to the fairly detailed and complex task of alternating between logic and content aspects of the lesson while the PLATO author selects a particular teaching logic, fills in the lesson parameters pertinent to the logic and enters the program in three separate steps.

Ease of Handling Subject Matter

Most of the languages discussed in this chapter are assembled online with the exception of PLATO which compiles a lesson by searching the author's parameter tape according to the teaching logic selected. Electric computer-controlled keyboards or teletypewriters are most often used as interface units for entry of programs, although CW II programs may be assembled either from punched cards or magnetic tape. Programs in MENTOR are stored on magnetic or paper tape, PLANIT on magnetic tape or disk, and CW II on magnetic tape, disk, or punched cards. PLATO and CW II authors are often faced with the additional task of providing the auxiliary instructional material required for the image projectors and audio tapes. TELCOMP programs and associated data may be entered on-line from a teletype or off-line by paper tape. APL/360 data may be entered off-line from punched cards or on-line from an electric typewriter. COURSEWRITER II provides authors the option of entering programs on-line or by punched cards that are assembled off-line. An advantage of having programs assembled on-line is the direct conversation allowed between author and computer. As soon as a statement is entered the assembler checks for typing and syntax errors and if any are found, error messages are immediately displayed.

Perhaps the main advantage of on-line authoring is the immediate feedback regarding the assembler's acceptance or rejection of a statement. This type of interaction permits an author to proceed to the next step only after the assembler accepts the present step. Programs entered and edited off-line usually require a number of runs before achieving error free assembly. Since only one run is often performed each day, the off-line procedure could take a fairly long period of time.

It would appear, at first glance, that on-line program entry might take less time than off-line procedures because program editing may be carried out whenever an error condition is encountered. However, a number of CW II authors have found difficulty in obtaining enough on-line authoring time to enter their programs. They discovered that programs can be quickly punched on cards, assembled off-line, and then debugged on-line. However, this procedure won't work too well for remote terminals. Perhaps an investigation should be conducted regarding the possibility of permitting remote users to construct their programs off-line, on a device such as magnetic tape, and quickly input the program upon sign-on.

All languages that offer on-line authoring facilities also provide debugging features which permit authors to enter either the edit mode or the execute mode. Basically, the edit mode is used by authors to display, replace, delete, or insert program material. A few of the languages incorporate unique editing features. For example, CW II provides the capability to move a statement or group of statements from one section of the program to another. It also allows the contents of return registers, buffer, counters, and switches to be pre-set and then displayed before and/or after execution. MENTOR, PLANIT, and PLATO all offer editing features somewhat similar to those of CW II. Although APL does not contain CW II's move command, it permits editing portions of program statements. Thus an APL author may edit sections of a statement rather than replacing the statement.

A debugging feature, the execute mode, permits CW II, MENTOR, PLANIT, and PLATO authors to take the student's role. In this mode an author may execute entire programs, portions of programs, or even single statements to test lesson logic, material presentation or correctness of coding. Authors debugging BASIC or TELCOMP programs are somewhat restricted due to the fact that entire programs must be executed.

APL offers authors a powerful mode which, in a sense, combines both edit and execute. The author, once he has constructed his program, begins execution. If an error condition arises in APL the line number, statement, location in statement where error was encountered, and a brief error message is typed. Then, the system releases control to the author. At this point, the author may do some "trouble shooting" by asking for displays of variables and calculation of various expressions. Once the error has been discovered, the line can be edited, and execution continued from that particular line. This procedure can be followed until the program is executed and at the same time, hopefully, debugged. Another feature of this APL mode is that a program may be interrupted during execution. Once an interrupt occurs, control is transferred to the author who may specify a branch to another location in the program. Once the branch is specified APL resumes control, branches to the statement indicated and continues execution from that particular statement. In this way APL authors can be quite selective in the program sections they desire to execute.

Another unique APL editing feature is it's "trace" routine. Often authors find that their programs are executable but produce results that are in error. The author, if unable to find the error in logic, may put on a trace. During execution, the trace indicates line numbers executed and the results associated with each line. Thus, the author may then follow the logic of execution and perhaps find the error in an easier fashion than attempting to "hand pick" his way through a program listing.

Only one of the CBI languages, PLANIT, fulfills requirements of authors desiring a language which offers both numeric and non-mumeric powers. COURSEWRITER II, MENTOR, and PLATO are primarily non-numeric languages while APL, BASIC, and TELCOMP are almost exclusively numerically oriented.

MENTOR, like CW II and PLATO, lacks computational power, but provides author facility for writing programs which allow student-computer interaction in natural English. The pedagogic strategies used can be described as conversational tutorial dialogues by the system. Courses programmed in MENTOR include exercises for medical students in medical diagnosis and for business students in management decision-making.

The initial aim in development of PLANIT was to produce an author-language flexible enough to enable a lesson author to design and build instructional material in statistics. However, PLANIT evolved into a powerful language that could be used for both numeric and non-numeric purposes and has been utilized in presenting instructional material ranging from elementary school spelling and vocabulary to an undergraduate course in economics.

APL, BASIC, and TELCOMP were primarily designed for use by students and are often referred to as student computational languages. These languages provide highly interactive computing power and because of their simple arithmetic notation are quite easy to learn. Thus it is possible for students to design, code, enter, debug, and execute their own programs.

Finally, an important point in evaluation of CBI languages concerns the amount of time required by an author to produce a program equivalent to one hour of student terminal time. Program preparation, of course, refers to activities such as gathering and preparing lesson material, planning strategies, coding, entering, and editing the program. The authoring time required to perform relevant programming activities depends upon a number of factors. These factors include the author's previous programming experience, familiarity with material presented, types of logic and interface devices used, etc.

It is difficult to ascertain which language requires the least amount of authoring time per hour of student terminal time. Programs already produced have employed a large number of diverse logics and have been used to present a wide range of subject matter. Perhaps one method of comparing CBI languages, regarding programming time, would be to select competent authors from each language and have them prepare a series of specific programs using a different logic with each type of program. Unfortunately, such a comparison is feasible only if a research team has access to all languages. At present, this type of research study could be considered next to impossible.

An examination of the literature reveals that documentation regarding authoring time is very sparse. This may be due to the fact that until recently most of the CBI installations have been used for research purposes. However, indications are that preparation time for one hour of computerized instruction ranges from 75 hours (Swets & Feurzeig, 1965) to more than 300 hours (Bernstein, 1967). The average preparation time was conservatively estimated by Rogers (1968) to be at least 100 hours.

Ease and Power of Response Analysis

One aspect of non-numeric CBI languages that differentiates and perhaps shows their superiority over the numeric languages is in the domain of answer processing and types of answer-matching routines provided. All languages incorporate the exact answer match routine but CW II, PLANIT, PLATO, and MENTOR also feature auxiliary routines which allow the author to accept responses which vary in predictable ways from specified lesson answers. All four of the above mentioned languages feature a keyword matching routine which matches each word in the student's response against the author's specified answer. A keyword match is recorded only if the response contains keywords in the same order as specified by the author. PLANIT and CW II permit an option which will disregard keyword order, while CW II also allows the author to specify a further option which will record a keyword match provided that at least one of the words embedded in the student's response matches a word in the listed answer. Both CW II and PLATO have routines that estimate similarity of characters in the student's response with the specified answer. The minimum percentage required for a spelling match in PLATO is fixed at 80% while in CW II the author can easily use available functions to specify the minimum number or percentage of character matches required before a response is regarded as acceptable. COURSEWRITER II also incorporates a function which will mark the student's response and return a display of the correct portion of the response.

PLANIT and PLATO permit a student's calculated numeric response to be matched with the result calculated by the computer. Authors using CW II, PLANIT, or PLATO can also specify that a numeric response is acceptable if within specified limits of the actual answer. PLANIT provides two features not available in CW II, PLATO or MENTOR; it can perform a phonetic match in searching for words that sound alike and also check an algebraic response to see if it is mathematically equivalent to the listed answer.

Answer matching routines such as the calculated numeric match and numeric match with limits are easily implemented in the numeric CBI languages. However, it would be fairly difficult for an author, without a mathematical and linguistic background, to write functions which perform many of the other types of answer matching.

Every CBI language offers some form of branching which, on the basis of the student's response and/or previous performance, modifies the course presentation for the student. PLANIT and CW II are the only languages which provide an implicit branching procedure in the answer analysis, to control course flow; although, MENTOR provides alternate branches to be taken if questions are repeated. All languages feature conditional branching to provide for individualized sequencing of material. The most dynamic and viable conditional branching is provided by APL, PLANIT, and TELCOMP. APL enables an author to specify, in a single statement, many condition sets accompanied by an equivalent number of entry locations. Thus, when APL executes a branch statement, it creates the possibility of branching to one of many locations depending on the condition set that is true. On the other hand, PLANIT's decision frame is used to specify branching conditions based upon combinations of variables such as latency, errors, help received, time taken to execute various segments of the lesson etc.

Maintenance of and Accessibility to Student Accounting Information

All four author-languages discussed in Chapter II incorporate procedures for recording certain aspects of each student's performance as he progresses through an instructional session. COURSEWRITER II, MENTOR, and PLATO automatically update the student's performance records when he signs off. PLANIT, however, will not add a student's record for a particular session to his cumulative performance record unless he indicates to the system that he has "finished" the session. Performance recordings are maintained on magnetic tape by both PLATO and MENTOR and on disk or magnetic tape by CW II and PLANIT. Course-authors are permitted, by PLATO and PLANIT, to display student records on-line. PLATO further provides each author with the facility to request a graphic trace of any student's progress or to retrieve and review certain aspects of the recordings. An off-line printer supplies performance listings for CW II and MENTOR authors.

Occasionally, the contents of counters and switches are used by CW II authors to specify certain branching conditions. This presents the author with the problem of anticipating and keeping a running account of branching conditions, relevant data kept in counters, significance of switch status, contents of buffers and return registers, etc. The job of maintaining scorekeeping documentation, coupled with CW II's primitive single condition branch statements, adds greatly to the time taken to plan and code the lesson. On the other hand, PLANIT and PLATO access student performance records to obtain relevant data for branching conditions rather than relying on the contents of counters and switches. This ability to reference performance recordings certainly enhances an author's power to make numerous conditional branches based on the student's past and present performance.

Control Over Various Types of Interface Devices

A basic question that can be asked in evaluating CBI languages pertains to the designed use of these languages: Does each language enable an author to present students with diversified instructional material as would be presented in the classroom, seminar room, laboratory, library, etc.?

Obviously none of the present CBI languages considered fully meet these instructional aspects, but COURSEWRITER II perhaps comes closest. COURSEWRITER II was originally designed to be used primarily by teachers wishing to program diverse subject matter for students of various age and ability levels. One of the outstanding features of CW II is that it permits presentation of material through a variety of media such as the audio unit, CRT, and image projector and permits the student to use the instructional keyboard or light pen for response input. This means, for example, that these interface devices combined with specially designed dictionary characters and graphic displays can be used in helping teach a foreign language, grade one students to read, physics students to understand Ohm's law, etc. Presently, lack of computational power is regarded as the major drawback of CW II. Once CW II has a computational feature such as APL/1500 added to its repetoire, then this language will become a much more effective tool for instructional purposes.

APL/360, BASIC, PLANIT, MENTOR, and TELCOMP rely almost exclusively upon either a teletypewriter or a computer controlled electric typewriter as the student-subject matter interface. This type of interface is advantageous for users desiring remote terminals. However, these terminals make a certain amount of noise and may present a serious noise factor if a number of terminals are located in the same room. A review of CBI literature indicated no available research related to the effect of machine noise upon rate of learning.

Another terminal feature requiring consideration is terminal speed in displaying information. Wodtke (1967, p. 321) indicated that typewriters and teletypes are quite slow in displaying text thus rendering them as inefficient interface devices for highly verbal students. Therefore, if the lesson requires rapid textual displays, the CRT might be a more effective display device than the typewriter or teletype.

APL/1500, CW II, and PLATO provide authors with control of the keyboard, CRT, and image projector. PLATO, however, does not enable easy change of dictionary characters or graphic displays as does CW II.

Because of the types of interface devices used, users of these three languages are not faced with problems of noise and slow display speed as are users of the languages mentioned above. COURSEWRITER II, in addition to the interface devices mentioned, controls a light pen and an audio unit. Thus, CW II currently furnishes authors with control over the largest number of auxiliary interface devices. However, the variety of interface devices does not necessarily mean that CW II is the best available CBI language for presentation of material. Travers (1964) concluded that if a student was required to attend to a diverse range of sensory stimuli, the realism provided by multi-media presentation would hinder rather than aid information transmission. He was in favor of a student being exposed to only one type of sensory stimuli at any specific time. On the other hand, Glaser, et at. (1964) indicated that a wide range of multi-media presentation is not necessarily detrimental. The problem appears to be one of finding the proper "mix" of interface devices to facilitate learning for each student.

In review, there are a number of desirable and undesirable features associated with each language. The following list contains a series of CBI language features deemed desirable from an author-instructor point of view:

1. Provision for isolating the instructional logics from the content of instructional sequences. (PLATO)

2. A calculation mode, with powerful computing ability, which could be used by both author and student. (PLANIT)

3. Easy author access to student performance records. (PLANIT and PLATO)

4. Lesson building information available to authors upon request. (PLANIT)

5. Control over a wide variety of interface devices. (CW II and PLATO)

6. Powerful branching. (APL, MENTOR, and PLANIT)

7. A variety of conditional branching, based upon direct referencing of student performance records. (PLANIT and PLATO)

8. On-line and off-line author facilities. (CW II)

9. Simple arithmetic operations which enable use of common mathematical notation. (APL)

10. Easy debugging of logic. (APL)

11. Editing features which enable authors to edit portions of statements. (APL)

III. Needs of CBI Teacher-Authors

Until recently, formal education systems such as schools and universities relied predominantly upon book publishers and companies specializing in production of various educational supplies to provide them with necessary curriculum material. However, since CBI is a relatively new concept, school systems wishing to implement computerized instruction find their former sources do not provide required CBI curriculum material. One reason for lack of CBI material is the proliferation of languages and systems currently in use. Hardware, software, and instructional programs used by one system are usually incompatible with those of other systems. Thus, any curriculum material produced for one language would be useful to only a small subset of all CBI users.

In the near future, it is likely that quality curriculum material will be produced by book publishers and companies specializing in production of CBI materials; but this situation will probably not materialize until additional systems are available and more research and testing has been carried out with present languages (Rogers, 1968). Another contribution that would facilitate mass production of curriculum materials is the development of a common CBI language(s) which could be used on a number of teaching logics.

Since availability of CBI curriculum materials from outside agencies is negligible, the only other promising source of CBI materials is the classroom teacher (Rogers, 1968). In the last section, it was noted that the teacher-author requires a minimum of 100 hours to construct a program equivalent to one hour of student terminal time. It is quite improbable that teachers would be able to find enough time in their teaching schedules to construct one hour of CBI material per week. A proposal of this sort would presumably be rejected by classroom teachers because of the seemingly unrealistic amount of program preparation time required, and also by school administrators due to reasons of economics and logistics.

Therefore, the problem is one of finding ways to reduce the amount of time required to prepare programs. Programming time could possibly be reduced if: (*a*) the author was not required to spend a great deal of time learning the language; and (*b*) the author could write programs without the requirement that he possess the combined qualities of a curriculum specialist, learning theorist, programmer, typist, artist, etc.

In order to reduce the amount of time spent by a teacher-author in producing CBI material, there appears to be an urgent need for either a new language or at least an enhancement of present languages. If a new language is developed, it would necessarily be a more natural language than present languages and, hopefully, more meaningful and easier for authors to learn. A desired feature of this language would be the separation of course logic from course content. This type of course separation would enable authors to have the logic portion of a lesson designed by a team of curriculum specialists, behavioral psychologists, experienced programmers, etc. Teachers wishing to prepare a lesson would be able to select a desired pre-programmed logic and enter only the required content. Thus, this new language would not only reduce the amount of authoring time, but also provide a varied array of programs to be used for extensive research and testing into learning strategies and the interface "mix" required to provide optimum learning environments in various subject areas.

CHAPTER V

DESCRIPTION OF THE STUDY

An IBM 1500 Instructional System was installed at the University of Alberta during the early months of 1968. In the ensuing period of time a number of disadvantages have been noted with CW II as a CBI language. The most prominent drawback noted was the amount of time required to construct a course. Authoring times ranged to 600 hours for every hour of student terminal time. Author opinions were varied but indications were that, in addition to the amount of time required to adequately learn CW II, an excessive quantity of time was needed in planning, coding, entering, debugging, and documenting programs.

Based upon experiences with COURSEWRITER as an authoring language and the needs of CBI authors, as described in the last chapter, a new computer language was designed. The first section of the chapter provides the rationale for a new language while the second, and final section, presents a comprehensive description of the new language.

I. Rationale

As stated in the previous chapter, there is a need for a suitable authoring language primarily designed for teacher use. This means that CW II should either be augmented or be replaced by a completely new authoring language. If CW II is to be augmented, by adding various enhancements, the basic nature of the language could not be changed without modifications to the 1500 System or at least a rewriting of the

CW II assembler. Modification of the 1500 System would be, in a programming sense, a large undertaking requiring the services of experienced 1130 Assembler language programmers. Rewriting the assembler would present problems with core allocations and the resulting liason with the time sharing system. Otherwise, programs would need to be run as batch jobs on the IBM 1130 Computing System, which forms the base for the IBM 1500 Instructional System. Batch mode refers to complete execution of one program before the computer begins execution of the next program in a particular set of programs.

Because the 1130 System is quite small, programs can only be processed with the 1130 System completely dedicated to that particular "background" job. A background job is defined as the automatic execution of lower-priority programs when higher-priority programs are not using the system resources. This means that while programs are processed on the 1130 System, the 1500 System could not be used for any other activities. A large number of these background jobs would likely require extra operating time and probably force the hiring of an extra computer operator. Thus, augmenting CW II would be an expensive venture due to the lack of qualified 1130 assembler programmers, excessive programming, and the extra system resources and personnel required to process programs.

To overcome these difficulties, the author concentrated on the feasibility of designing a new CBI authoring language. The new language, named VAULT (a Versatile AUthoring Language for Teachers), was based upon the following three major areas stated in chapter IV: (a) desirable criteria determined by reviewing other CBI languages; (b) desirable features from other CBI languages; and (c) current needs of CBI teacher-authors. Once designed, a subset of VAULT was developed and implemented to test the feasibility of the total language. It should be noted that the basic design of VAULT was occasionally modified as unanticipated authoring needs were encountered. This was especially true when VAULT was used in training a group of teachers to prepare CBI materials.

The initial aim was to separate VAULT into a logic division and a data division. The VAULT logic division was designed to be used by experienced VAULT programmers who also had some background and experience in COURSEWRITER. This division corresponds to a normal concept of a computer program as in any other computer language (i.e., FORTRAN). It was envisioned that each logic program would be prepared by a team of instructional specialists consisting of educational psychologists, learning theorists, curriculum specialists, philosophers, etc. Material prepared by these specialists would then be programmed by experienced VAULT authors. Once researched and documented these "programmed logics" would be documented and added to a library of VAULT logics. The teacher or subject matter specialist would select a suitable programmed logic from a pool of available logics and, through the VAULT data division, enter subject matter to compliment that particular logic. An important point is that the logic program forms the basic framework to which parts of data must conform.

By separating logic and data the study endeavoured to provide a means whereby: (*a*) the time taken to learn VAULT would be less than that taken to learn COURSEWRITER; (*b*) teacher authoring time would be effectively reduced; (*c*) pre-programmed logics would provide educators with a powerful research tool for testing various learning theories and instructional strategies; and (*d*) the data division would provide the teacher with a quick method of entering subject matter.

Both the logic and data divisions of VAULT accept two types of entry records. The *Division Control Record* (DCR) is used in both divisions while the *VERB* record is used only in logic and *KEYWORD* record in data. Three types of DCR's used by VAULT are: (*a*) major DCR's which are necessary for organizing or subdividing a course into formalized instructional domains; (*b*) minor DCR's which are required for internal organization and modification of the course domains; and (*c*) leading DCR's which supply information to the VAULT compiler.

The logic author would use major DCR's primarily to define hierarchical levels or instructional domains within a course. The domains, in descending hierarchical level are BLOCK, LESSON, and PROBLEM. Any course programmed in VAULT usually subsumes several BLOCKS, each BLOCK contains a series of LESSONS, and each LESSON, in turn, is composed of a number of PROBLEMS.

The domain of a BLOCK is analogous to the group of specific instructional methods or techniques utilized by a classroom teacher to achieve educational objectives that are related to a particular unit or chapter of a course. A varying number of LESSONS would usually be required to obtain the desired objectives for each BLOCK. The domain of a VAULT LESSON could be analogous to a single classroom period in which a particular instructional sequence is used by the teacher in attempting to teach a specific concept.

Contained within the boundaries of each LESSON domain are a series of contingent interactions between the student and the computing system. The domain (level) at which these interactions take place is called the PROBLEM. Basically, logic instructions are entered at the PROBLEM level.

The three domains of a VAULT produced course (BLOCK, LESSON, and PROBLEM) are important to both the logic and the data authors. First, let us examine the implications of these domains to the logic programmer. Further to providing the means for logically subdividing a course the major DCR's would automatically initiate a series of system defaults. These defaults would be activated within any PROBLEM that requires a student response. If activated (enabled), system defaults would: (a) specify parameters for conditional branching based upon the student's performance only within that particular PROBLEM; and (b) include automatic accumulation of student performance data.

When a BLOCK level is initiated by a major DCR that particular BLOCK would acquire default options identical to those specified for the entire course. If the logic author decides that he does not wish conditional branching to occur when a student has "timed out" a specific number of times, within any PROBLEM subsumed by the BLOCK, he would merely enter a minor DCR that overrides and disables the default related to timed out conditions. It is also possible that within the same BLOCK the author desires to enable the timed out condition for a specific LESSON. If so, the author would immediately follow the LESSON major DCR by the minor DCR that modifies timed out conditions. Included in the minor DCR would be conditional parameters to be used in evaluating timed out conditions within any PROBLEM contained in the boundaries of the LESSON domain. Thus each hierarchical level obtains its defaults from the next higher level within the domain. However, any default could be overriden or modified at any of the three hierarchical levels. Thus the author could accept the systems defaults or modify them to meet the particular requirements of the course.

The VERB is the second type of record accepted by the logic division. This record permits an author to use meaningful action words to indicate points in the logic program where specific computer units are to be used and associated material from the data division is to be accepted and/or to control the logical flow of the program. The use of VERBS is restricted to the PROBLEM level.

VERBS which specify use of particular computer units would have names similar in meaning to the author actions necessary to activate the unit. For example, the teacher would DISPLAY material on the CRT, ERASE the CRT, SHOW a film on the image projector, PLAY a recorded message, etc. Designation of the computer unit is not a difficult task. However, specifying positional parameters with these types of VERBS could prove troublesome, especially to the beginning logic author. Therefore, a number of VERBS that require parameters would be supplied with special default parameters. This VERB default feature would permit the beginning author to write simple logic programs without need of knowing about the associated VERB parameters.

Another worthy VERB characteristic would be the IF (condition) --- THEN (action) conditional statement. The author could easily control the student's path through a course by specifying conditional action based upon the student's response to a question or upon certain aspects of his performance in the course (i.e., IF WRONG THEN GO TO NEWPR). The author would also be provided with a VERB (i.e., STOP) that enables him to specify execution of a course to be terminated. (IF COUNTER-15<8 THEN STOP).

VAULT attempts to provide the logic author with some mathematical capability by allowing him to ASSIGN numerical values to variables through use of mathematical expressions. These variable names could be used to replace parameters associated with any logic record. When VAULT processes records containing variables, the variables would automatically be restored to their numeric equivalents.

The VAULT data author should find entry of subject matter easier to learn and simpler to use than that of the COURSEWRITER author. A primary reason, of course, is that the data author begins with a pre-programmed logic that forms the basic framework for the method in which data is to be entered. The data DCR's are not as complicated to use as those in logic because no system defaults are involved. For the most part, major DATA DCR's are used only to subdivide subject matter into BLOCK, LESSON, and PROBLEM. These data domains, upon entry, would be matched against those in logic. However, the data DCR's provide an author with some control over use of certain portions of the pre-programmed logic. Occasionally the data author finds that the amount of subject matter, to be used in a program, is either more or less than anticipated by the logic author. He would then use the major DCR's to control movement from one logic domain to another. Initially the first three DCR levels in data (BLOCK, LESSON, and PROBLEM) are synchronized with the first three corresponding levels in logic before processing of data begins. As each PROBLEM is encountered in data VAULT would skip to the next PROBLEM in the logic LESSON domain. If the last PROBLEM in the current logic LESSON has already been passed, VAULT would then return to the first logic PROBLEM within the LESSON domain and begin a second pass through that logic LESSON. Thus VAULT would continue iterating sequentially through the PROBLEMS within a logic LESSON until a LESSON is encountered in data. When a new LESSON is encountered in the data material then VAULT would automatically skip to the next logic LESSON and continue matching logic and data.

Similarly, if a new LESSON is encountered in data but the last logic LESSON, within the current logic BLOCK, has already been passed VAULT would return to the first logic LESSON in the BLOCK and continue processing data. If an end of data record is encountered then VAULT would skip the remainder of the logic program and stop any further processing.

The second type of data record, called the KEYWORD, provides the teacher with a method for entering the subject matter or course material. Some of the KEYWORDS must correspond to the VERBS in the logic division. Thus, if during processing, VAULT encounters a VERB that indicates textual material is to be displayed on the CRT (i.e., DISPLAY) then the next data record must contain a specific conformable or synchronous KEYWORD (i.e., TEXT). It is important that each conformable KEYWORD be meaningful and, as well, synchronous with the VERB to which it corresponds in logic. Non-conformable KEYWORDS, on the other hand, do not conform with any VERBS. These KEYWORDS are associated with instructions which specify entry of student responses. They permit the teacher to easily enter complete lists of answers and, also, reinforcement messages.

A number of important features, embedded in the data division, permit the teacher-author to quickly and easily enter data material. The VAULT data "text editing" feature represents a significant improvement over the tedious and time consuming coding method required by COURSEWRITER authors. The data author would merely enter textual material in free form and let VAULT edit and format the text according to screen parameters from the corresponding VERB in logic. In addition, as VAULT generates code to display the textual material it also examines the VERB parameters to determine which portion of the screen is to be erased and when indefinite pauses are to be inserted. Thus the VAULT data author uses a quick one step method to enter textual material.

A second important feature is that the VAULT data author can string together a complete list of answers of one type (i.e., CORRECT). This could be accomplished by entering the answer list on the non-conformable KEYWORD record that denotes the answer classification (i.e., CORRECT, WRONG, etc.). To accomplish the same effect a COURSEWRITER author would need to enter single COURSEWRITER statements to test each answer included in the VAULT answer list.

Also, the VAULT author does not need to supply each answer with a unique answer analysis identifier as does the COURSEWRITER author. Each answer in the data answer list is automatically supplied with an identifier by VAULT. Further, VAULT checks the system default options, available at the current logic PROBLEM level, and generates the required score keeping statements.

Another feature of VAULT data is that a teacher could immediately follow a particular answer list by a reinforcement message. VAULT would examine the parameters of the VERB requesting the student response, determine the portion of the CRT allocated for a reinforcement message, edit the textual material, and produce code to display the message. If the teacher does not include a message he could then request VAULT to display an appropriate "system" reinforcement message. VAULT would maintain a series of reinforcement messages that correspond to particular answer types (i.e., WRONG, UNREC, CORRECT, etc.). Upon request from the teacher-author, VAULT would randomly select a message from a group of messages corresponding to the type of answer and then display this message in the CRT area designated by logic VERB which requested the student response. To ensure that each student has ample opportunity to read the reinforcement message an indefinite pause would be generated along with a system message which informs the student that he may proceed by pressing the space bar.

Since the construction of VAULT logic is quite independent of the data division VAULT permits logic material to be compiled without necessarily being followed by data input. This means that the logic author would be given the opportunity to debug logic material before it is made available to a data author. However, since much of the data material must synchronize with the logic material VAULT does not permit data to be compiled and processed unless first preceeded by an error free logic program.

VAULT programs would be run in batch mode on the IBM System 360, Model 40 or larger. Output from the logic division would be integrated with the teacher's input to the data division, by the System 360, and COURSEWRITER source code would be the final product. The COURSEWRITER source could then be transferred to the 1500 System for program assembly.

There were a number of reasons for selecting the System/360 over the 1130 System. The anticipated difficulties with implementing an augmented CW II language in the 1130 System were discussed earlier in this section. Similar difficulties likely would arise if VAULT were to be used with the same system. Also, as more programs are written and the 1500 System used mainly by students, authors will probably find their on-line authoring time reduced. In fact, authoring time will probably be restricted to program debugging.

On the other hand, institutions using the 1500 system will probably have access to an IBM System/360. The System/360 offers high speed, larger core, and a greater number of peripheral storage devices than does the 1130 System. On this basis, it was decided that VAULT would be processed by a pre-compiler written in PL/1, which is a high level programming language designed to serve both scientific and commercial programming needs. Statements written in VAULT would be punched on cards, run as a batch job on the System 360, and then COURSEWRITER source code would be output and assembled on the 1500 System.

Since VAULT programs are to be run off-line, on the System 360, it is necessary that course authors be provided with options for obtaining comprehensive program listings. Different sets of listings would, of course, be required from the logic division and from the data division. The logic author would need a listing of the logic input cards and, as well, a listing of the output code that is passed on to the data division. Included in the latter listing would be a number of tables containing information to help the logic author debug his program. These tables would provide a complete inventory of items such as alphabetic labels used, the input statement number where the label occured, the level at which the label occured and whether or not branch statements were made to that label. If branch statements are encountered in the logic input, a table would be displayed indicating the input record number containing the branch statement and the label to which it refers. These features could prove quite beneficial to the logic author in his efforts to either expand upon the logic program or just in debugging and editing the input material.

The data division would provide the teacher with a listing of the data entry records and also a listing of the COURSEWRITER source code produced by VAULT. Since the amount of COURSEWRITER code could be fairly extensive the latter listing would separate the code according to its correspondence with the data LESSON levels. The listing would be designed so that each LESSON level would begin only at the top of a new page. Included at the top of each LESSON page would be a message indicating the beginning of a LESSON and, in addition, a statement which indicates the number of the data LESSON entry record that corresponds to this LESSON of COURSEWRITER code.

If any errors are encountered, by VAULT, during processing of logic or data material then error messages would be included in the listings. Each error message would be listed immediately following the record at which the error occurs. All errors which occur would be classified into one of three levels of severity.

The lowest error level would merely notify an author of a specific error, the system would assume certain default conditions, and processing would continue normally. If the second level of error is encountered this would be severe enough to suppress either logic or COURSEWRITER output depending, of course, upon the VAULT division in which the error occurs. However, VAULT would continue syntax checking all input records for that division only. When the most severe level of error occurs VAULT would immediately terminate further processing of input records.

A final note is that the VAULT compiler will be "table driven." This means that all major terms and messages and decisions used by VAULT will be incorporated into a series of tables. Most of the terms would be basically teacher oriented, but it is possible that teachers will find a few terms confusing. If so, these terms could be easily replaced by more meaningful terms. Also, as the language is expanded and augmented, new terms could be added to the tables.

Since the compiler is table driven it should be possible to convert the English terms to their counterparts in other national languages (i.e., French, German, etc.). In this way VAULT could be made available to teachers working with similar CBI installations in other countries.

II. Description of VAULT

Basically, VAULT was designed with major objectives being to produce a meaningful, teacher oriented language which separates logic and data. The desired outcomes were to reduce learning time by teachers, decrease coding effort, increase production of CBI courses, and increase research activities related to CBI. However, designing a new CBI language was one problem, but being able to demonstrate the operational feasibility of the language was, of course, a very different problem. In order to demonstrate the feasibility of VAULT, a decision was made to code only a subset of VAULT. The emphasis in determining the subset was to select sections of VAULT which beginning authors could easily use to produce most types of courses currently written by COURSEWRITER authors.

The remaining sections of this chapter have been included to give a description of the following facets of VAULT: (*a*) organization of VAULT; (*b*) logic division; (*c*) data division; (*d*) correspondence and internals of the two divisions; and (*e*) listings produced.

Organization of VAULT. VAULT is separated into two main divisions: LOGIC and DATA. It is the logic division which sets course strategy, selects interface devices, and specifies types of student accounting records to be maintained. The data division primarily provides the teachers' course material or subject matter.

Because VAULT is divided into two sections, it provides educational researchers with a vehicle for researching learning theories and also provides teachers with a quick and easy method of subject matter entry. Persons using VAULT would not necessarily need to know data processing techniques since VAULT uses natural, meaningful words to convey what the author wants done. Those persons possessing only a very general knowledge of computer concepts could easily learn to use VAULT.

It was intended that the logic division would be programmed by experienced authors while the data division would be used by all types of authors, particularly teachers. Logic programmers would use VAULT to effect various teaching strategies that have been thoroughly researched and tested. It is assumed that VAULT logic programs would be formulated by teams of educational researchers. The programmed logics would then be documented and made available to teachers. A teacher wishing to produce instructional material would first select a suitable available logic. Then he would use the data division as a means of entering desired subject matter. Next, the two divisions would be integrated by the IBM 360/67. Final output from the System 360 would be in COURSEWRITER II code which, in turn, would be used as input to the IBM 1500 System. Perhaps, at this point, it should be noted that the data section is dependent upon the logic program for its logical structure. This means, therefore, that some portions of the data section are necessarily conformable to the logic section specifications.

A principal type of instruction contained in both divisions is the Division Control Record (DCR). The DCR's are divided into three groups: leading, major, and minor. The leading DCR is similar in concept to a part of job control language used in conjunction with other computer languages. It is the leading DCR's which enable an author to supply instructions to the VAULT compiler regarding production or suppression of various listings, COURSEWRITER source code, assembly cards, etc. Major DCR's could be used to subdivide either logic or data into three hierarchical levels or subsets called BLOCKS, LESSONS, and PROBLEMS. The PROBLEM represents the lowest level or smallest subset within these two divisions. Usually, although not always, a PROBLEM may be associated with a student response. Each LESSON may contain up to 99 PROBLEMS and each BLOCK may include 99 LESSONS. Finally, the total COURSE may consist of 99 BLOCKS. The intent of the LESSON was that it would represent the logical structure associated with presentation of a particular concept or perhaps be the equivalent of a single "classroom period." Being the largest subset, the BLOCK could perhaps be analogous to a course unit or textbook chapter. Minor DCR's in the logic section supply specifications regarding strategies within any subset of BLOCK, LESSON, or PROBLEM.

Logic division. Any logic input is classed either as a division control record (DCR) or as an action record (VERB). In general the DCR and VERB in the logic provide the logical course structure which is later complemented by subject matter from the data division. In order that logic authors not confuse the two records, all DCR's are prefixed by two dollar signs (i.e., $$LOGIC).

A. Division control records (DCR's) are divided into the following categories: (*a*) major DCR's--$$LOGIC, $$BLOCK, $$LESSON, $$PROBLEM, $$ENDLOGIC; (*b*) minor DCR's--$$LATENCY, $$TIMEOUTS, $$CORRECTS, $$WRONGS, $$CWSTART, $$CWEND; and (*c*) leading DCR's--$$LIST, $$LOGICNAME.

These DCR's were designed to enable the programmer to:

1--signify the program beginning with the DCR $$LOGIC and program ending with the DCR $$ENDLOGIC. Each card containing these DCR's may contain comments. The end of logic file, or $$ENDLOGIC, denotes that all further logic input will be ignored.

```
EXAMPLE:  $$LOGIC     BEGINNING OF MULTIVARIATE ANOVA
              .
              .
              .
              .
          $$ENDLOGIC  END OF MULTIVARIATE ANOVA.
```

2--subdivide the logic section into subsets containing hierarchical levels of BLOCKS, LESSONS, and PROBLEMS. These levels are formed by the major DCR's $$BLOCK, $$LESSON, and $$PROBLEM. Each of these DCR's may be accompanied by a unique alphanumeric label of six characters or less. Labels permit references or branches to be made to any labelled major DCR from any point in the program.

```
EXAMPLE:  $$BLOCK    STATS1
          $$LESSON   CORREL
          $$PROBLEM  LINEAR
```

3--specify system default options that are to be in effect for BLOCK, LESSON, or PROBLEM levels. Minor DCR's are important because they provide the author with means for modifying system defaults or options concerning student responses. VAULT defaults would be in effect for an entire course, however they may be overridden for any subset by use of minor DCR's.

The following VAULT system defaults specify action to be taken regarding student responses:

LATENCY--The maximum latency default for any keyboard or lightpen response is 90 seconds. If the latency time is exceeded, the response would be considered as a timeout.

CORRECT--Normally the student would be passed to the next PROBLEM if a correct answer is entered.

TIMEOUT, WRONG, UNRECOGNIZABLE--If any of these three conditions is encountered the student would be permitted to retry answering the question. However, if any single condition is encountered a total then the student would be taken back a maximum of three PROBLEMS for review. The system would not branch a student back past the beginning of the current LESSON.

System defaults may be overridden by using minor DCR's such as $$LATENCY, $$TIMEOUTS, $$CORRECTS, $$WRONGS, or $$UNRECS immediately following any of the three major DCR's which set up the hierarchical structure. Each minor DCR will effect a change in the system action for all PROBLEMS at that level or any immediately following levels that are lower in the hierarchy. For example, if a minor DCR is encountered after a $$LESSON then its options will apply for all PROBLEMS within that LESSON. It is also possible to use further minor DCR's with different options to modify system defaults for any PROBLEM within that LESSON.

The procedure for deciding which system defaults and associated modifications will be in effect for each level could be accomplished as follows. When a $$BLOCK is met the VAULT system defaults (discussed above) are passed from the COURSE level down to the BLOCK level and will remain in effect for that entire BLOCK. Next, the VAULT compiler checks for immediately following minor DCR's and modifies the BLOCK defaults if any are found. Whenever a $$LESSON is encountered, the BLOCK defaults are duplicated at the LESSON level. Again, any minor DCR's following a $$LESSON would modify the defaults only for that particular LESSON.

Note that any logical level within the hierarchical structure always receives its basic default options from the last processed higher level. Thus it is possible for an author either to omit minor DCR's and simply rely on system defaults throughout the course or he may carefully combine major and minor DCR's to produce sophisticated default branching.

An explanation and example of use of these minor DCR's is listed below.

```
$$LATENCY 75
```

This record specifies that all questions occuring as a subset of the level containing this minor DCR shall have a latency period of 7.5 seconds. The latency parameter is expressed in tenths of seconds. Normal system latency time is 90 seconds.

```
$$TIMEOUTS  YES,6,2
$$WRONGS    YES,6,2
$$UNRECS    YES,6,2
```

These three minors have been grouped together because the resulting action is similar. For example, the above examples specify that a student will be branched back a number of PROBLEMS (i.e., 2), for review purposes, if the individual cumulative count of either TIMEOUTS, WRONG, or UNRECOGNIZABLE answers in any single PROBLEM reaches a particular maximum (i.e., 6).

```
$$TIMEOUTS  NO
$$WRONGS    NO
$$UNRECS    NO
```

The single parameter NO specifies that no individual cumulative count is to be kept of answers which were timed-out, wrong or unrecognizable. The system action will merely be to permit the student to retry the question if a correct answer is not entered.

$$CORRECTS YES,25,20

Since VAULT permits only one question per PROBLEM, this minor DCR actually operates at the LESSON level. When this minor is enabled, VAULT will accumulate the number of correct responses within a LESSON. Once the total number of correct answers within the LESSON reaches a particular value (i.e., 25) the system will branch the student ahead according to the number of PROBLEMS specified by the parameter (i.e., 20). Note that if there are not enough PROBLEMS left in the current LESSON to accommodate the desired branch, then the student will be branched to the beginning of the next LESSON. The usual system action for a correct answer is to skip the student on to the next PROBLEM.

$$CORRECTS NO

This record specifies that no cumulative count of correct responses is to be maintained. If a correct answer is encountered the system continues on to the next PROBLEM.

4. Modify student performance recordings. One feature of VAULT is that various counters, switches, buffers, and return registers are reserved to maintain student accounting information. The system will maintain a running total of the number of problems, attempts (not including timeouts), timeouts, corrects, and wrongs or unrecognizable answers in each BLOCK and LESSON as well as in the entire course. In addition, count is maintained of the number of timeouts, wrong and unrecognizable answers entered in any PROBLEM.

By allowing the system to control these counters the author is relieved of some of the drudgery of specifying counters to be incremented, decremented, etc. If the logic author wishes to use a number of reserved counters for other tasks or perhaps does not wish to have particular counters altered by the system he would specify the parameter "NO" with any minor DCR (i.e., $$WRONGS NO). The counters affected depend upon the level at which these minor DCR's would be used. For instance, if $$CORRECTS NO is specified at the LESSON level, the counter reserved for accumulating the number of correct answers within a LESSON would not be cleared at the beginning of that LESSON. This could be useful to the author who wishes to test the number of correct answers obtained in the LESSON completed prior to the current LESSON.

In addition, VAULT will use return register 1 to store the label of the current data PROBLEM and return register 2 for the label of the last executed PROBLEM.

5--supply compiler information. The two leading DCR's $$LIST and $$LOGICNAME provide information to the VAULT compiler. Information as to whether or not the author desires a listing of the logic records entered and/or a listing containing the label table, go to table, variable table, and table of object code produced by the logic division would be included in the accompanying parameter. In this way the author could ask for none, one, or both listings. If the $$LIST record is omitted only a listing of logic entry records would be provided.

Each course generated by the logic program requires an alphabetic name containing five characters or less. The record $$LOGICNAME would be used to enter a unique name. However, the author could omit this record and the system would default to the name VAULT.

6--provide facility for inserting COURSEWRITER II code. By using the minor DCR's $$CWSTART and $$CWEND it is possible for the logic author to enter actual COURSEWRITER II source cards. The $$CWSTART record indicates the beginning of the COURSEWRITER cards while the $$CWEND record denotes the end of COURSEWRITER source (Appendix D, p. 180). These two DCR's enable an author to enter certain COURSEWRITER capabilities currently not possible with the VAULT subset.

B. VERBS or action words perform two basic functions: (a) to provide means for accepting subject matter from the data division; and (b) to control the logic structure of the COURSEWRITER II program which is finally produced by VAULT. The combination of DCR's and VERBS provides the underlying logic which is to be used in conjunction with the data division.

VERBS used to accept subject matter from the data section are:

DISPLAY
QUESTION
POINT
SHOW

These verbs indicate locations at which the teacher-author would provide subject matter. Also, each verb contains sets of parameters to be used for controlling entered data. The logic author would use either the system default parameters or enter specific parameters to suit his particular needs.

The VERB DISPLAY (Appendix D, p. 182) indicates that the data section must provide textual material. Accompanying parameters, if given, specify the portion of the screen to be reserved for entry of subject matter from the data division. If no parameters are given, VAULT would then default to the uppermost 28 rows of the CRT. Since the verb DISPLAY denotes the CRT will be used for exhibiting textual material, VAULT first erases the screen of any previously displayed material. However, if the author wishes to retain a particular CRT display and further display subject matter on a designated portion of the CRT he would do so by adding the modifier INSERT and a few associated coordinates to the record containing the DISPLAY verb. For example, the instruction DISPLAY INSERT 10,5,10,20 specifies that the portion of the screen beginning at row 10, column 5 and extending for 10 rows and 20 columns should first be erased and then reserved for subject matter from the data section. In addition, the author could affix a label to the DISPLAY record. The label could be used in later logic statements to refer to the set of coordinates associated with the DISPLAY record.

If textual material entered by the teacher exceeds the screen area specified by the DISPLAY parameters then an indefinite pause would be generated. Thus the student would be given the opportunity to control the pace at which more than one screen of textual material is to be presented. Note that the system does not generate an automatic pause for a DISPLAY instruction unless the subject matter entered, in the data division, exceeds the screen area specified by the parameters associated with the verb DISPLAY. A logic programmer who desires to control the length of time a particular set of material is to be displayed would do so by entering the verb PAUSE along with the parameters.

Any coordinates in the parameters specified could be represented by variable names. Care must be taken to provide display coordinates that lie within the allowable limits of the screen.

The format for the verb QUESTION (Appendix D, p. 185) is similar to the verb DISPLAY. QUESTION provides the author with the means to present the student with a question, accept a keyboard response, and display a reinforcement message. Also, by use of the modifier INSERT, it is possible to specify particular parameter coordinates or to use system default parameters as well as to affix a label.

Instead of a single set of parameters as used by the verb DISPLAY, three sets of parameters are associated with QUESTION. The first set of paramters specifies the CRT area to be used in displaying question material; the second set of parameters refers to the CRT area where student responses would be entered; and the third set of parameters refers to the portion of the screen where reinforcement messages would be displayed. An author, instead of specifying the QUESTION parameters, could default to the system parameters. VAULT defaults to 10 rows for textual material, 4 rows for the response, and 6 rows for the reinforcement message.

The verb POINT (Appendix D, p. 187) is used to enable teacher-authors to specify conditions for a light pen response. It is similar in format to both DISPLAY and QUESTION but does not allow for use of the modifier INSERT.

The POINT record includes two sets of parameters. The first set of parameters refer to the CRT area where the teacher could pose a question and the second set of parameters refer to the CRT area where reinforcement messages could be displayed. If any parameters are omitted then system defaults would be used.

Usually the verb POINT is preceeded by a number of DISPLAY INSERT records each accompanied by a label unique to that problem. Labels are associated with display coordinates which define various areas where light pen responses could be made. The teacher-author could use these logic labels to indicate various areas of the screen containing answers which are correct, wrong, etc.

To indicate use of the image projector a logic author would enter the verb SHOW (Appendix D, p. 188). This verb provides the parameters necessary for controlling presentation and sequencing of filmed material. Three optional parameters could be used by the author. Proper sequencing of these parameters permits specific film frames to be located, the shutter to be opened or closed, and, if opened, the amount of time the shutter is to remain opened.

A group of VERBS which require no corresponding entry from the data section are ERASE, PAUSE, IF---THEN, GO TO, STOP, and ASSIGN. Some of these VERBS also permit the logic author to control logical flow of the COURSEWRITER II program that is output as the final product of VAULT. A brief description of each of the above mentioned verbs is included in this section.

The logic programmer would use the verb ERASE (Appendix D, p. 184) to erase all or any portion of the CRT. Usually ERASE is accompanied by a set of two coordinates although it is possible to omit the coordinates and default to the system coordinates. Omission of the ERASE parameters causes the entire screen to be erased. If a parameter set is specified it could contain the coordinates for the starting row and the number of rows to be erased. It is possible to use a DISPLAY label to represent the specified area to be erased. The label would provide VAULT with the screen coordinates. It should be noted that the label used as an ERASE parameter would have previously been defined in the particular problem in which it is used.

The verb PAUSE (Appendix D, p. 185) could be used by authors to insert specified pauses which would effect action during execution of the COURSEWRITER II program. The author could include a single parameter defining length of pause in tenths of seconds or may omit the parameter entirely. Omitting the parameter would cause execution of the COURSEWRITER program to be halted indefinitely and a message displayed which would direct the student to "press the space bar to continue."

When the verb GO TO (Appendix D, p. 188) is encountered, VAULT treats it as an unconditional branch statement to be inserted into the logical flow of the COURSEWRITER program. Each GO TO instruction must be followed by a label. This label must appear at any of the BLOCK, LESSON or PROBLEM levels, must be alphanumeric, begin with an alphabetic character, and must be less than seven characters in length.

The verb IF---THEN is contained in two separate entry records and specifies that if a certain condition is true (i.e., IF CORRECT) then the following action would be carried out during student execution of the program (i.e., THEN GO TO NEXT01). With the current VAULT subset, the IF condition could be associated with counters, switches, types of answers, time taken to respond, etc.

An author could use the STOP verb to halt a student's progress through a course. During execution time the student would be branched to the end of that block. Also, this VERB could be used as the indicated action in an IF---THEN conditional statement (i.e., IF WRONGS>=25 THEN STOP).

Occasionally the verb ASSIGN (Appendix D, P. 191) is called a compile time verb because it is used only during logic compilation and is not directly used by the data section. The verb ASSIGN allows an author to assign numeric values, within the range -32768 to +32767, to variables. These variables must be alphanumeric, begin with an alphabetic character, and can not exceed six characters in length.

Assigned numeric values could take the form of mathematical expressions which use either the add, subtract, multiply, or divide operators. However, any expression used in an assign statement may not contain more than one operator although several assign statements could be made in a single ASSIGN record (i.e., ASSIGN A=2 X1=Y-5). If the author wishes to delete any variable name from the file containing variables he would do so by using the verb ASSIGN followed by the variable name minus any numeric assignment to that variable. For example, the record (ASSIGN A=A+2 X1=3 Y) assigns numeric values to variables A and X1 and deletes Y from the file of variable names.

DATA Division. Entry of data is accomplished through use of either a Division Control Record (DCR) or a keyword. A number of data DCR's have similar names to and perform functions quite similar to those in the logic division. Keywords, on the otherhand, perform functions which are very different from those of VERBS in logic.

Each keyword ends with a colon because a few of the keywords have names similar to the VERBS in logic. Use of the colon was intended to permit authors to conceptually keep keywords separated from VERBS and thus not confuse their functions while alternating between logic and data.

A. Division Control Records (DCR's) in the data division, as was the case in the logic division, are classified into the following types: major, minor, and leading. Major DCR's are used to signify the beginning and end of the data records and to subdivide the subject matter into BLOCKS, LESSONS, and PROBLEMS. The minor DCR's supply various editing features while the leading DCR's provide VAULT with necessary information for use in preparing COURSEWRITER output for assembly on the 1500 System.

The major DCR $$DATA signifies the beginning of the teachers' data records. Any information accompanying these two DCR's would be treated as a comment. Content material or subject matter is divided by the DCR's $$BLOCK, $$LESSON, and $$PROBLEM in a manner similar to that in the logic division. However, no information from immediately following minor DCR's is incorporated as would be the case in the logic division. The main function of the latter three major DCR's is primarily to group subject matter in units that could easily be handled by the teacher.

No BLOCK may contain more than 99 LESSONS and, in turn, no LESSON may contain more than 99 PROBLEMS. Any labels associated with the data DCR's $$BLOCK, $$LESSON, and $$PROBLEM would not be used by the compiler. These labels, however, would be produced in the COURSEWRITER listings as an aid for the teacher in identifying various subdivisions of the subject matter entered.

Only two minor DCR's, both associated with editing of textual display materials, are available for use by the teacher-author. The first minor DCR, $$ROWSIZE, would be used to specify spacing of text on the CRT. Normally the system uses two CRT rows for any simple line of text and does not attempt to place any blank rows below that line of textual material on the screen. However, the teacher could specify that a number of blank rows, up to a maximum of 26, be inserted below each line of text.

The parameter used in conjunction with $$ROWSIZE is numeric and is determined by using the basic two rows required to display a simple line of text and then addir the required number of blank rows to be inserted following each line of text. For example, $$ROWSIZE 4 specifies that two blank rows are to be inserted below each line of text to be displayed on the CRT.

In order that textual material is presented in the portion of the screen designated by parameters of the verb DISPLAY in the logic division all text entered by the teacher is edited by VAULT. This means that occasionally some words at the end of a display line must be hyphenated. The teacher could maintain some control over hyphenation of words through use of the minor DCR $$HYPHEN. If this record is not entered by the teacher VAULT would not attempt to hyphenate any words less than six characters in length. However, if the $$HYPHEN record is encountered VAULT would, if necessary, hyphenate any word which has at least as many characters as specified by the HYPHEN parameter. When any word is hyphenated VAULT places at least the last two characters of the hyphenated word on the next display line.

Leading DCR's in the data section would be used to provide VAULT with all information necessary for production of instructions required to assemble the COURSEWRITER source program on the 1500 system. These DCR's include $$AUTHOR, $$COURSENAME, $$CWDECK, $$LOGPK, $$STARTBLOCK, and $$STARTLBL. To obtain COURSEWRITER source code either in punched cards or on tape the author would usually include the first four leading DCR's and their associated parameters. If the data to be entered represents an addition to an already existing BLOCK then the author would include $$STARTLBL. The parameter on the record $$STARTLBL indicates the label, in the existing COURSEWRITER program, after which the additional program is to concatenated. When the teacher wishes to assemble a new BLOCK of COURSEWRITER code he would include a $$STARTBLOCK record containing the number of the new BLOCK.

B. Keywords represent means by which a teacher-author would enter subject matter for any COURSEWRITER program produced by VAULT. The current VAULT subset contains the keywords TEXT:, QUESTION:, TIMEOUT:, CORRECT:, WRONG:, ADDITIONAL:, UNRECOGNIZABLE: or UNREC:, and SHOW:. Note that all keywords end with a colon. The colon was included as part of the VAULT keywords in an attempt to remove any confusion that could perhaps arise from the correspondence and similarity between keywords in data and VERBS in logic.

Four major uses of keywords are: (a) to provide curriculum material; (b) to provide a method for requesting and accepting student responses; (c) to specify various answer lists to be used for comparison with student responses; and (d) to provide correctional or reinforcement messages.

Basically, each record used as an entry to VAULT could use any portion of a punched card from column 1 through 71. Thus any material punched in card columns 72 through 80 would be disregarded by the VAULT compiler. The above condition holds for any entered record whether it be in logic or data.

Since the logic section contains the basic logical structure for production of COURSEWRITER there are certain VERBS in logic which require specific keywords to be entered at corresponding locations in the data section. Keywords required at specific locations in data are called conformable keywords. The two conformable keywords in data are TEXT: and QUESTION:.

The keyword TEXT: provides a quick and easy method for entry of textual material that is to be presented on the CRT. Each TEXT: record conforms to and must maintain the same relative location within a data PROBLEM as the corresponding DISPLAY verb within a logic PROBLEM.

An author who wishes to display text ranging from a single character to an entire "chapter" would probably do so by placing the desired material on as many cards as required providing that the first card of that group of text is headed by the keyword TEXT:. The VAULT compiler would read the material on the TEXT: card and any immediately following cards until an end-of-text or "enter" symbol is encountered. The enter character is produced by using the 0-2-8 key on the IBM 029 cardpunch. In all examples of VAULT code the enter symbol will be represented by "□".

This means, of course, if the set of text designated as a single display cannot be completed within the first 71 columns of the initial card containing TEXT: that the remaining text would be punched in a continuous stream on as many cards as required. When all material related to the preceeding keyword TEXT: has been entered, the author would then conclude the material by concatenating an "enter" character to the text. Therefore any material contained between TEXT: and the enter symbol would be edited and formatted by VAULT to fit the CRT area designated by the coordinates of the corresponding DISPLAY in logic.

Since all VAULT programs are currently entered through use of punched cards the format for any special characters used by the 1500 System are described in the 1500 CAI Programming Systems Reference Card. There are, however, four special characters used in data entry that are unique to VAULT.

These special characters have been included to aid the teacher in editing and debugging textual material. The character "*" indicates, to the compiler, that any material that follows must be displayed on the next CRT display line; "¬*" specifies that the remainder of the card is to be disregarded and material from the next card is to be displayed on the next CRT display line; "¬$" denotes that the remainder of the card is to be disregarded, an indefinite pause inserted into the COURSEWRITER source, the display area erased during execution time, and material from the next card to be displayed at the uppermost line of the display area; "¬#" indicates that the compiler should disregard the remainder of the card and continue "reading" material from the next card in a manner similar to the case when the 71st column of a text card is normally processed.

Thus through the use of these four special sets of characters the teacher could exercise control over placement of text on the CRT. Also, it should be emphasized that any time the amount of textual material exceeds a designated display area VAULT inserts an indefinite pause into the COURSEWRITER source.

The second conformable keyword, QUESTION:, normally heads a series of non-conformable keywords used in response analysis. Depending upon the corresponding VERB in logic, each QUESTION: and its associated keywords perform one of the following two functions: (*a*) display a question, accept a keyboard response, supply sets of criterion answer lists, analyze the response, and supply a reinforcement message; or (*b*) display question material, accept a light pen response, supply labels corresponding to the CRT locations of various types of answers, and display a reinforcement message.

When considered separately from the other keywords, the QUESTION: record permits a teacher to ask a question or provide instructions to the student before he makes his response. The format for the QUESTION: record is the same as that for TEXT:. The first card used with this type of record contains the word QUESTION:, followed by the question material or instructions, and ended by the enter symbol. As many cards as required to present the material could be used because the VAULT compiler keeps processing data until an enter symbol is encountered. If the amount of question material exceeds the question portion of the CRT, defined by coordinates of the corresponding VERB, then an indefinite pause would be generated automatically.

Immediately following the processing of any QUESTION: record, VAULT expects to encounter at least one non-conformable keyword. A non-conformable keyword does not have any VERBS in logic with which it is associated. These keywords would be used by teachers to provide information for use in the analysis of student responses.

Non-conformable keywords currently available to the teacher-author are TIMEOUT:, UNREC: (or UNRECOGNIZABLE:), CORRECT:, WRONG:, and ADDITIONAL:. The former two keywords in the above list provide a data author with the facility to enter specific reinforcement messages if the student either exceeds the latency time specified for the question or enters a response not included in the teacher provided list of anticipated responses. The format for these two records is exactly the same as that for TEXT: and QUESTION: (i.e., TIMEOUT: <Y>OU HAVE TAKEN TOO LONG TO ANSWER.<T>RY AGAIN.□).

An author who wishes to provide a different message each time the student enters an unrecognizable answer, within any single problem, would do so by providing a number of UNREC: records each containing the desired message. These records would necessarily be entered in the same sequence as the anticipated presentation of messages associated with each record.

Note that UNREC: records are usually entered in sequence following the other non-conformable keywords. This sequencing requirement is necessary due to the fact that once the section of COURSEWRITER response analysis which deals with unrecognizable answers is encountered, the student's answer is then considered as being unrecognizable..

One important facility provided by VAULT is the generation of appropriate reinforcement messages. If the author indicates that he wishes a system generated message VAULT would randomly select a message from files of messages reserved for each non-conformable keyword. To obtain a system reinforcement message for either a TIMEOUT or UNREC condition the author would need only to omit entry of the associated keyword records.

To suppress reinforcement messages for TIMEOUT and UNREC conditions, the author needs only to enter the associated keyword followed by a space and an enter symbol (i.e., UNREC: □). These two characters, in effect, instruct the VAULT compiler that no message is to be displayed.

Whenever a reinforcement message is displayed VAULT automatically produces an indefinite pause and a message, at the bottom of the CRT, which instructs the student to press the space bar when he is ready to continue. This feature was included to insure that each student has an opportunity to read each message and could control the pace at which he moves on to the next portion of the course. However, if the author suppresses the reinforcement message no pause would be generated and the course execution would proceed.

The other three non-conformable keywords are <u>CORRECT:</u>, <u>WRONG:</u>, and <u>ADDITONAL:</u>. In addition to providing the same facilities for displaying reinforcement messages as the keywords TIMEOUT: and UNREC: these three keywords permit the teacher to enter sets of anticipated correct, wrong or additional answers. The format for these keywords differs from the format for TIMEOUT: and UNREC:. When using any of these three keywords, the author would immediately follow the designated keyword either by a set of answers, if a keyboard response is to be entered, or by a list of TEXT or DISPLAY labels if a lightpen response is to be entered. The reinforcement message would follow the answer list.

Each answer or label in the teacher supplied list would necessarily be followed by a semicolon with the last answer or label being immediately followed by an enter symbol (i.e., "WRONG: YELLOW;BLUE;VIOLET□<T>HAT COLOR IS NOT INCLUDED IN THE LIST.□"). Also, it is possible to enter a number of records which use the same keyword. This permits the author to provide unique messages for particular types of answers. Thus, in the previous example listed, the author could have entered three different WRONG: records each with a specific message for the accompanying wrong answer(s).

If the author wishes to accept any response and consider it as either correct, wrong, or additional then he would enter a record containing the desired keyword followed by an enter symbol and message. For example, if a QUESTION: record is immediately followed by the record "CORRECT: □<C>ORRECT.□" then the following would happen when a student passes through corresponding portion in the COURSEWRITER program--(a) if the student exceeds the latency time then a system produced TIMEOUT message would be displayed because the author omitted the TIMEOUT: record; or (b) if the student enters a response it would be considered as a correct response and the message "Correct." displayed along with the system message "Press Spacebar to Continue".

In the above example any student response would be considered correct because the CORRECT: record was encountered first in the input sequence and was not accompanied by an answer list. However, if that particular CORRECT: had been preceeded by the record "WRONG: □<T>RY AGAIN.□" then regardless of the answer entered by the student it would be considered a wrong answer.

In each of the latter two examples a reinforcement message was included following the specified answer set. Actually, the method for indicating a particular type of reinforcement message is much the same as that for TIMEOUT: and UNREC: with the exception being that message specification must follow the answer set rather than the keyword. Thus, to indicate display of a teacher supplied reinforcement message, the desired message would follow the first enter symbol (i.e., WRONG: COLUMBUS□<N>O, HE SAILED IN 1492.□); to request an appropriate system message the first enter character would be immediately followed by a second enter character (i.e., WRONG: COLUMBUS□□); and to suppress the display of any reinforcement messages the first enter symbol would be followed by a space and an enter symbol (i.e., WRONG: COLUMBUS □ □).

The non-conformable keyword ADDITIONAL: has format requirements similar to that of CORRECT: and WRONG: with the exception that VAULT does not generate system messages for an additional answer. There is a difference, however, in the method of answer analysis during execution of a COURSEWRITER program. For example, if the student's response is included in the set of answers included on a particular CORRECT: record then that response is judged correct and execution continues according to specifications in the logic division. On the other hand, if the response is included in the set of answers listed on an ADDITIONAL:

record then a specific counter is assigned a numeric value according to the relative position of that answer within the list. This means that if a response matches the fifth answer in the ADDITIONAL: answer set then a designated counter would be assigned the value 5. Thus, in using the ADDITIONAL: record, the teacher would first need to become familiar with documentation of the logic program regarding the effect of particular positioning of answers or labels within that record.

The keyword SHOW: is optionally conformable in that it may be omitted even though the corresponding verb SHOW exists in the logic section. When the SHOW: record is included in data it would contain a single parameter specifying the frame number of the film to be presented (i.e., SHOW: 835). If this record is omitted VAULT would use the frame number supplied in the logic section.

Correspondence and Internals of Both Divisions

Basically VAULT consists of two divisions or sections called LOGIC and DATA. The logic division first accepts high level input from a logic programmer and maintains a file of VAULT "object code" which contains a description of the basic logical structure of the course. This basic logical structure consists of a series of "logic vectors" sequentially built up as material is entered into the logic division.

Upon encountering the DCR $$ENDLOGIC, VAULT logic makes a single pass through the file of logic vectors and checks all branch (GO TO) statements for unresolved labels. If no serious errors are encountered during logic processing then the file of logic vectors would be stored on disk. Also, a listing of this particular disk file could be obtained through use of the logic DCR $$LIST. A description of this listing is included in the next section.

Once the logic section has wrapped up its activities, by storing the logic file on disk, procedures would be initiated for entry of material to the data division. When the major DCR $$DATA is encountered the data division then retrieves groups of logic vectors from the file on disk. This file dictates, to a large extent, the type of conformable keywords acceptable for entry at particular points in the data section.

Therefore, material entered in the logic section could be compiled without necessarily being followed by subject matter input from the data section. The converse, however, does not hold true. The data section needs to be immediately preceeded by associated logic material input before any teacher-author data could be entered.

Logic vectors corresponding to BLOCK, LESSON, and PROBLEM levels each contain information regarding system and author specified default conditions. Author specified defaults were indicated by minor DCR's immediately following any major DCR in logic. A series of logic vectors, each corresponding to a separate VERB entry record, follow each PROBLEM logic vector.

The teacher-author, in using a pre-programmed logic, would have the option of supplying exactly as much subject matter as indicated by the logic or of skipping or repeating certain portions of the logic program. If the course represents an attempt to obtain experimental information the teacher would, perhaps, enter the exact amount of data required by the logic program. However, it is nearly impossible for persons designing logic programs to anticipate the amount of subject material a teacher desires to supply. Therefore, VAULT permits a teacher-author to skip or repeat certain portions of the programmed logic.

The teacher, in designing data entry records to correspond to the logic material, would begin by matching major DCR's (i.e., $$BLOCK, $$LESSON, and $$PROBLEM). Actually, the data DCR $$LESSON is the key which permits a teacher to modify amounts of data material to be entered and thus skip or repeat portions of logic. During normal entry of data records VAULT sequentially proceeds through the file of logic vectors and matches logic and data. However, only when a $$LESSON is processed in data would the next LESSON level in logic be used. This means that if the end of a current LESSON is encountered in logic but a new $$LESSON is not processed in data, VAULT would return to the first PROBLEM within that logic LESSON and continue processing data records.

For example, Fig. 1 (p. 111) illustrates the correspondence between two PROBLEMS in logic LESSON ONE and three PROBLEMS in data LESSON SOC. Notice, however, that as soon as a $$LESSON is encountered in the data section VAULT "wraps up" activities associated with the current logic PROBLEM, skips the remainder of the logic LESSON, and initiates action to begin processing the next LESSON in both divisions. Therefore, the number of data PROBLEMS entered within any data LESSON could be less than, equal to, or greater than the number of PROBLEMS contained within the corresponding logic LESSON.

In addition to being able to control the number of data PROBLEMS, within any given lesson, the teacher could also control the number of data LESSONS within any BLOCK. The resulting action is much the same as that for controlling the number of data PROBLEMS. Again the key to skipping or repeating LESSONS within any given BLOCK is the data DCR $$LESSON.

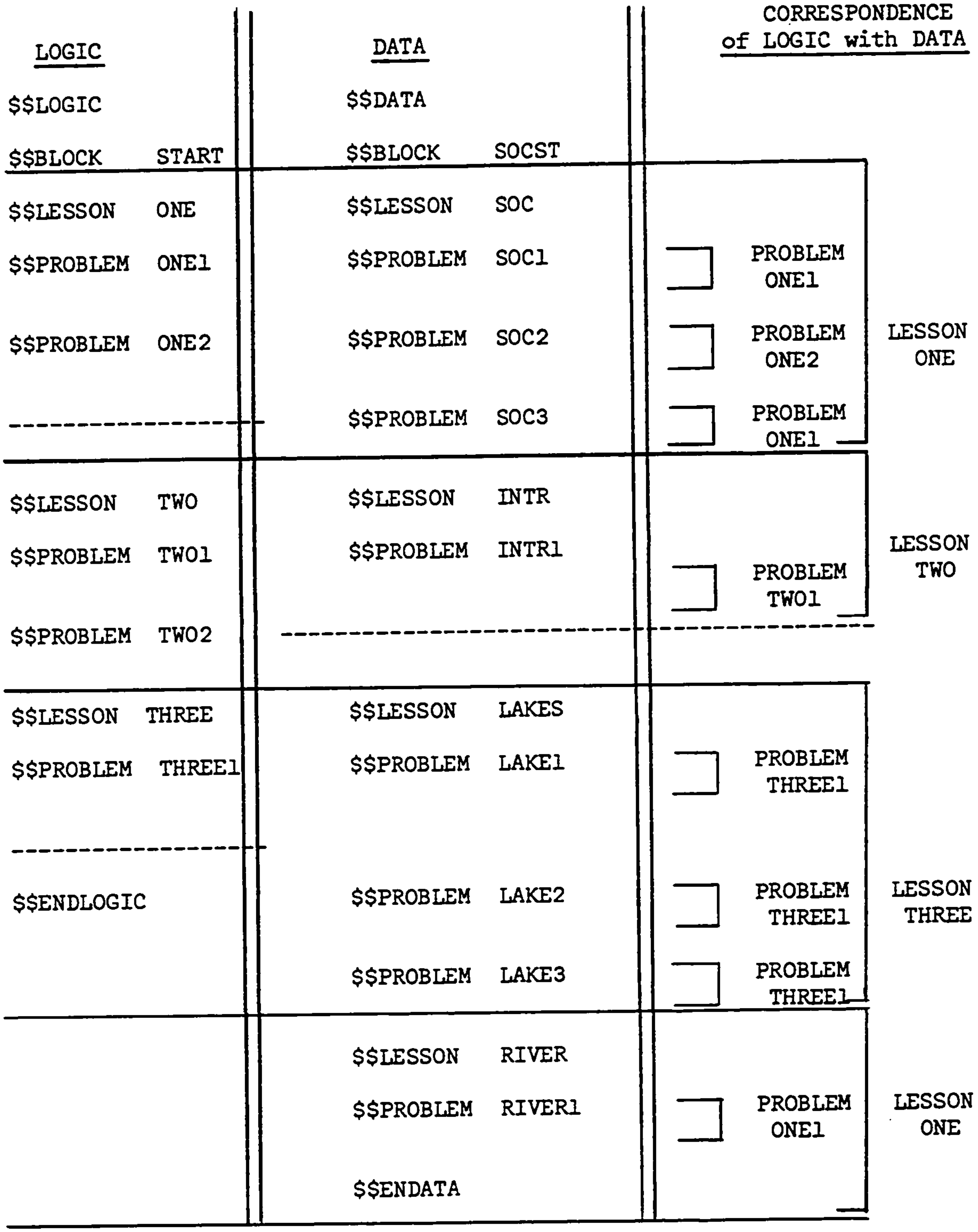

Figure 1. Major Levels of Correspondence Between Logic and Data

When the condition exists that a $$LESSON is encountered in data and the last logic LESSON has been processed VAULT normally returns to the beginning of the logic vector file and repeats the first logic LESSON. The $$ENDATA record, on the other hand, signals VAULT to complete the current PROBLEM and stop further processing regardless of the number of logic LESSONS used. Thus the number of data LESSONS within a BLOCK, as is the case with number of data PROBLEMS within a LESSON, could be less than, equal to, or greater than the number of logic LESSONS within any BLOCK.

The remainder of this section presents a detailed description of methods and decisions used by VAULT in integrating the logic vector file and the subject matter entered in the data section. The final product, of course, is COURSEWRITER source code.

When the data DCR $$BLOCK is processed VAULT initiates its basic labelling procedure. All major data levels, whether BLOCK, LESSON, or PROBLEM are supplied with a unique six character numeric label within the course. The first set of two digits represent the number of the current BLOCK, the next two digits the number of the current LESSON within that BLOCK, and the last two digits the number of the current PROBLEM within that LESSON. For example, the label 052683 could be read as the eighty-third PROBLEM of the twenty-sixth LESSON of the fifth BLOCK. Therefore, in examining the COURSEWRITER listing produced by VAULT, the author could examine the most recent label and easily determine the particular BLOCK, LESSON, and PROBLEM.

Once the BLOCK level is labelled (i.e., 010000) VAULT checks the first logic vector to ascertain which system defaults and student accounting records are to be maintained throughout that particular BLOCK.

Appropriate counters and switches would then be cleared. A similar procedure would be carried out when the first data $$LESSON is processed.

It is at the PROBLEM level that a major portion of COURSEWRITER source is produced. When the first $$PROBLEM within a LESSON is encountered VAULT merely moves on to the first logic vector contained in the current LESSON and produces COURSEWRITER code which at student execution time would initiate a restart point (i.e., PRR), clear various PROBLEM level counters for use in student accounting, and load the labels of the last executed PROBLEM and the current PROBLEM into return registers. When this particular initiation routine is finished VAULT would begin sequential processing of the logic vectors within the currect LESSON.

If the current logic vector contains a VERB which requires a conformable keyword in data the next data record would be processed. The first word on this record would be extracted and a check made to determine if it is the required keyword. For example, if the logic vector represents the verb DISPLAY then VAULT would examine the first word on the next data card and check for the keyword TEXT:. If the keyword TEXT: is sensed VAULT would check the display logic vector for an accompanying label and, if one is found, store the CRT display coordinates and label in a special display file. This file would be available for use by the verb ERASE or to supply coordinates for use in the analysis of light pen responses. Next, VAULT would sequentially process data records until an enter symbol is encountered. All textual material would be edited and formatted according to the logic display parameters. Thus, once the processing of a PROBLEM has begun, it is necessary that all required data be available and conform to the PROBLEM logic since currently it is not possible to skip or repeat any portion of a logic within a PROBLEM.

In the above example, if the conformable keyword TEXT: is not encountered in data, while VAULT maintains a "pointer" at the display logic vector, then a message indicating that "data is out of synchronization with logic" would be printed on the data input listing. All data cards to the next $$LESSON would be listed but no syntax checking carried out. When the next $$LESSON is encountered, in data, VAULT would normally move on to the next logic LESSON and resume processing.

Only the logic verbs DISPLAY, QUESTION, and POINT require conformable keywords to be processed in data. Upon encountering either QUESTION or POINT, in the logic vector file, VAULT checks the next data card in search of the conformable keyword QUESTION:. If the keyword is present COURSEWRITER source would be produced to permit a student response to be entered and processed. Included in this code would be a ten character response identifier consisting of the six character numeric label in the current COURSEWRITER PROBLEM and the initial four characters of the label associated with the QUESTION or POINT logic vector.

Any logic control verbs (i.e., PAUSE) which immediately follow the QUESTION (or POINT) logic vector but preceed the first IF---THEN vector would be converted to appropriate COURSEWRITER code and inserted following the code for the enter and process. The IF---THEN conditions contained in subsequent logic vectors, within the current PROBLEM, would be stored in files corresponding to TIMEOUT, CORRECT, ADDITIONAL, WRONG, and UNREC response groups. At this point, the teacher's QUESTION material and data records containing non-conformable keywords associated with the QUESTION would be processed. As each non-conformable keyword is encountered various counters would be modified to maintain student accounting information as directed by the defaults for that PROBLEM, LESSON, and BLOCK.

Each teacher supplied answer within a PROBLEM would be affixed with a sequential two character answer analysis identifier. The first character of the identifier would be alphabetic and would correspond either to the initial letter of the answer type or to the next letter in the alphabetic sequence (i.e., TIMEOUT-T&S; CORRECT-C&D; ADDITIONAL-A&B; WRONG-W&X; UNREC-U&V) while the second character would be alphanumeric (i.e., 1-9&A-Z). For example, the list of correct answers within a single PROBLEM would be sequentially identified by answer analysis labels beginning with C1 and ranging to DZ (i.e., C1,C2,C3,...,C9,CA,CB,... CZ,D1,D2,...DZ). This means that each answer grouping could contain 70 unique identifiers. If more than 70 answers are supplied for any one group (i.e., WRONG:) then the latter half of the identifiers would be repeated (i.e., X1,X2,...,XZ).

When the end of data record, $$ENDATA, is encountered VAULT begins its final "wrapup" procedures. A check would be made of the most severe type of error, if any, that occured during data processing. All errors encountered during data entry would be listed in the form of four character error messages (i.e., E213) immediately following the record at which the error was discovered. The first character of the message, either X, E, or W, indicates the severity of the error and the general action to be taken by VAULT; the last three characters provide a reference number, ranging from 000 through 999, to be used by the teacher in a "look-up" of the possible cause of error and appropriate corrective procedures to rectify the error.

An X-level (TERMINAL) error is the most severe type of error that could occur during record entry. When an X-level error is encountered further processing would be immediately terminated and no listing nor punched cards of COURSEWRITER source code would be provided.

The E-level (SEVERE) error would cause suppression of further COURSEWRITER code. However, syntax checking of input data records would continue. If this error occurs and the $$LIST parameter indicated a listing of COURSEWRITER code then a listing of code, up to the point in the program where the first E-level error occured, would be produced. Output of punched cards would be totally suppressed.

When a W-level (WARNING) error is encountered certain default features are assumed but processing continues normally. The W-level message was intended to warn teacher-authors that particular portions of the data input and statements in the COURSEWRITER source would be altered to meet logic vector requirements.

During final wrapup procedures VAULT checks parameters from the leading DCR's initially entered to the data division. These parameters, as well as severity of errors encountered during processing, determine whether none, one, or both of the following would finally be output: (a) a listing of COURSEWRITER source code; and (b) a punched deck or tape containing source code. If any one of the above conditions is fulfilled then VAULT automatically inserts an extra six-character numeric label at the end of each LESSON.

As VAULT produces a listing and/or punched deck of COURSEWRITER code each branch statement is first checked for unresolved labels. Unresolved branches likely would occur due to the teacher entering data which repeats or skips certain portions of logic. To resolve these statements the following procedures would be conducted: (a) if a COURSEWRITER branch statement refers a label that does not exist within an indicated LESSON then the statement would be modified so that the branch would be made to the last label within the indicated LESSON; and (b) if a COURSEWRITER branch statement refers to a LESSON that does not exist within the current BLOCK then the referenced label is inserted at the end of the BLOCK of code. Every COURSEWRITER statement that fulfills any one of the above conditions would be immediately followed by a W-level error message.

Listings Produced

Both the logic DCR $$LIST and the data DCR $$LIST enable an author to request different sets of listings regarding a particular VAULT program. The files and tables which comprise a VAULT listing could be easily identified by the following headings: (a) VAULT LOGIC DIVISION, (b) LABEL TABLE, (c) GO TO TABLE, (d) ASSIGN TABLE, (e) LOGIC OUTPUT, (f) VAULT DATA DIVISION, and (g) COURSEWRITER OUTPUT. The remainder of this section contains a brief description of each of the above files and tables.

A. The VAULT LOGIC DIVISION file simply prints a copy of every logic record entered in the logic division. This particular portion of the total VAULT listing could be obtained by including "YES" as the first parameter accompanying the logic DCR $$LIST.

Each statement in this listing is preceeded by a statement number representing the relative position of that card within the entire logic entry deck. The statement numbers were intended to aid the logic programmer during debugging stages since the four listings that follow occasionally refer to these statement numbers. Further, to enable an author to easily check logic input and quickly locate particular LESSONS or PROBLEMS, VAULT inserts a long dashed line before each $$LESSON statement and a short dotted line before each $$PROBLEM statement.

All error messages produce in VAULT logic, or data for that matter, would be listed in the following manner: "*** ERROR E 193 ***". An error message would always appear in the line directly below the statement to which it refers.

B. The LABEL table lists all labels entered in associated $$BLOCK, $$LESSON, and $$PROBLEM logic cards. Every BLOCK, LESSON, and PROBLEM level is assigned a VAULT generated six character numeric label regardless of whether or not the author includes an alphanumeric label as the parameter on the corresponding entry record.

Each statement contained in the label table is comprised of the following: (a) the numeric label, (b) the teacher provided label--it could be blank, (c) the DCR level (B, L, or P), (d) the statment number at which the record appeared in the VAULT LOGIC DIVISION listing, and the symbol "Y" if this label was referenced by a GO TO statement. Currently all labels are listed in the order they were entered. Once the label table is listed and all logic records have been entered VAULT would check all branch statements for unresolved labels. If any unresolved labels are encountered then, following the label table, a message would be printed listing the label and the statement number of the input record which contained the unresolved GO TO statement.

C. The GO TO table lists all author provided labels referred to in GO TO statements as well as the associated statement number in the listing of the VAULT LOGIC DIVISION file. This listing could be useful to the author who wishes to add new DCR levels and labels to the current logic program. Before entering a new label the author could check the new label name against labels listed in the LABEL table to guard against duplication of label names. VAULT does not permit duplication of logic labels due to the confusion that could arise by using branch statements which refer to one of these duplicate labels. Further, the author could also use this listing to locate every GO TO statement that refers to a particular label, especially if he desires the change that particular label name.

D. The ASSIGN table lists all variables specified by the author and used during compilation of the program. In addition to the variable name, the listing includes the numeric value of the variable at conclusion of compilation.

E. A file called LOGIC OUTPUT is the final logic listing produced by VAULT. This file, along with the previous three tables, could be obtained providing no serious error occurs during compilation. To request these listings the author would enter the word "YES" as the second parameter in the logic DCR $$LIST.

The LOGIC OUTPUT file contains a series of logic vectors. If no serious errors have occured, to this point, then this file becomes part of the output from logic and later is input to the data division. Usually this portion of the VAULT listing would be used only by experienced logic programmers who are familiar with the special code. All information contained in the logic vectors would have been checked and edited to provide the data division with all the information necessary for accepting data records to produce COURSEWRITER source code.

The format for this listing is similar to the listing of logic input records. A long dashed line is placed above the statement which represents each LESSON logic vector. Inserted between each dashed line and the LESSON logic vector is an expression indicating the statement number in the VAULT LOGIC DIVISION which contains the corresponding $$LESSON input record.

F. The VAULT DATA DIVISION file presents a copy of each teacher-author input record. A format similar to the one used in listing the logic input records would be used. There is no method by which the teacher could suppress this particular listing.

To aid the teacher in debugging his data, LESSON levels are separated two parallel dashed lines and PROBLEM levels are separated by short dotted lines.

G. COURSEWRITER OUTPUT is the final and, perhaps, the longest listing provided by VAULT. To obtain this listing the author would either omit the data DCR $$LIST and permit the system to use its default option to produce this listing or he could include the word "YES" as the parameter on the data DCR $$LIST. This listing could be suppressed by using "NO" as the parameter on the data DCR $$LIST.

Since there is usually a great deal more COURSEWRITER code produced than data records received VAULT begins each LESSON of COURSEWRITER code at the top of a new page. Each new LESSON page is headed by the word "LESSON" which is then followed by a VAULT statement indicating the data input statement number which contains the corresponding $$LESSON card.

CHAPTER VI

RESULTS AND EVALUATION OF VAULT

This chapter is composed of three sections--the first presents a resume of the history and background of VAULT, the second a detailed evaluation of VAULT, and the third a parallel between two existing COURSEWRITER programs and two similar programs written in VAULT. Included in the second section is an examination of: (*a*) the fulfillment of original objectives; and (*b*) the attainment of criteria for an evaluation of CBI languages, as stated in the first portion of Chapter IV.

I. Background

One of the major aims of this study was to design a new CBI authoring language which would better meet the needs of teachers who wish to produce computer-based instructional materials. A number of current CBI languages were reviewed and an attempt made, from a teacher-author point of view, to include desirable features from these languages into the design of a new language. This new language is named a *V*ersatile *A*uthoring *LA*nguage for *T*eachers.

The design phase of VAULT began in November, 1968, and was completed about three months later. Once basically designed, VAULT, theoretically, possessed many qualities necessary for a teacher oriented authoring language. The next step in determining the feasibility of VAULT was, of course, to develop and implement the language specifications. Because of the estimated time and effort required to code the entire language a decision was made to concentrate upon a subset of VAULT. This subset included the portions of VAULT which would enable teachers to easily program CBI instructional strategies such as drill and practice, tutorial, and simulation.

The development phase began in February, 1969, and was carried on for five months. VAULT was coded in PL/1 and programs were tested on the IBM System 360/67 at the University of Alberta. Finally, in July, 1969, the VAULT subset was ready for use by teachers and experienced COURSEWRITER authors.

During the month of July, 1969, a workshop to train teachers to program the IBM 1500 System was held at the University of Alberta. The workshop, sponsored by the Alberta Human Resources Research Council (HRRC), was conducted during a four week period. Each teacher who completed the workshop received an honorarium of 75 dollars. The group consisted of four elementary school teachers, four junior high teachers, two senior high teachers, one teacher from a school for retarded children, and a university professor. Subject areas taught by these teachers ranged through mathematics, science, and social studies in elementary school; art, social studies, and mathematics in junior high school; mathematics and physical education in high school; and music education in university.

The twelve teachers in the workshop were randomly assigned to two groups of six. One group, consisting of two females and four males, with median age 28 years, received instruction in VAULT while the other group, one female and five males, with median age 37 years, received instruction in COURSEWRITER. After two weeks, the groups were reversed and received training in the other language.

Following the workshop, each participant completed a questionnaire (Appendix B, p. 155). The questionnaire was intended to provide biographical data as well as information regarding the teachers' attitudes toward VAULT and COURSEWRITER. Also, as a portion of the implementation and evaluation phase this writer selected two diverse instructional programs, coded in COURSEWRITER, and constructed two parallel courses in VAULT.

II. Evaluation

The evaluation of many computer languages (i.e., APL, COBOL, FORTRAN, PL/1, etc.), as to whether or not they should be developed and made available to other institutions, certainly did not depend on any tests of statistical significance regarding the relative merits of various languages. Rather, each language was basically designed to meet the needs of major user groups and examined to determine how well it met previously formulated criteria. Periodically many of these "complete" languages are re-evaluated and then revised and/or augmented to meet the growing needs of the primary users.

Similarly, it would be somewhat meaningless to try to show any statistical difference between authors using VAULT and COURSEWRITER, especially in view of the fact that only a subset of VAULT was developed and implemented. The importance, therefore, is not to show that VAULT is a "better" or a "worse" authoring language than COURSEWRITER but, rather, to determine how well certain objectives were achieved.

Some data, included in Appendix C, were obtained from the HRRC teacher workshop. These data were intended primarily to provide descriptive information regarding teachers' criticisms regarding VAULT design concepts, difficulty in learning and using VAULT, and adequacy of the DATA manual provided. This information, along with a measure of teacher attitudes toward the VAULT subset was used as a part of the data for evaluating VAULT as a CBI language.

The major objectives of this study were: (*a*) to design a new, natural, high-level authoring language (VAULT) which would separate course logic from course content and use terms meaningful to teachers; (*b*) to develop and implement a subset of VAULT which could be applied to produce a wide range of teaching strategies similar to those already programmed and available; (*c*) to effectively reduce the amount of teacher time taken to learn and use VAULT, rather than COURSEWRITER, in producing CBI materials; (*d*) to appraise teacher-author attitudes towards VAULT; and (*e*) to draw conclusions regarding the feasibility of the total language from the results obtained within the subset.

With the exception of the latter objective, all these objectives are included within the six major criterion groups, listed in Chapter IV, and are used in evaluating VAULT as a CBI language. Therefore, discussion related to the above objectives is integrated within the following criteria: (*a*) meaningfulness; (*b*) extent of separation of logic and data; (*c*) ease of handling subject matter; (*d*) ease and power of response analysis; (*e*) maintenance of and access to student accounting information; and (*f*) control over various types of interface devices.

Meaningfulness

VAULT fulfills the prime requisite that a CBI language be meaningful to the needs and activities of its principal users. Although VAULT can be used by authors with varying degrees of programming experience, it is the data section that is specially oriented to the classroom teachers who now and then wish to program CBI materials but do not want to spend a great deal of time learning the language. In examining VAULT, it is noted that most instructions have connotations synonymous with terms related to ordinary classroom activities (i.e., LESSON, PROBLEM, QUESTION, CORRECT, etc.).

Though sophisticated logic programs would probably be written by experienced VAULT programmers the subject matter to compliment logic programs would most likely be provided by classroom teachers. In order to supply data material teacher-authors should be able to easily read logic instructions. This is achieved, in VAULT, through use of meaningful action words called VERBS. For example, the verb DISPLAY means textual material is to be displayed on the CRT, ERASE specifies that a portion of the CRT is to be erased or cleared, POINT indicates a light pen response is to be accepted from the student, etc. It should be noted that VAULT could be easily learned because natural, complete words are used to specify instructions to the VAULT compiler (i.e., IF WRONG THEN GO TO REMED1).

Meaningful data words, called KEYWORDS, are used rather than the abbreviations or operation codes used in COURSEWRITER. To enter subject matter a teacher needs to learn and master only a very limited number of KEYWORDS.

Also, as an aid to help teachers quickly and easily learn VAULT, concise error messages are provided at three levels: warning, severe, and terminal. Each error message contains an explanation of the probable cause of error and specific corrective procedures to be taken. The warning level assumes certain default conditions and modifies the program according to these defaults. A severe error causes suppression of COURSEWRITER source code but syntax checking of data input continues. If more than one error is encountered in a single input record then unique error messages would be listed below the statement in error.

Examination of results of the HRRC workshop shows that, in the initial two week period, the group trained in COURSEWRITER learned to display textual material on the CRT, partially learned to accept keyboard responses and performed simple response analysis. The VAULT group, on the other hand, learned to use the VAULT data section and were able to display textual material on the CRT after the first day of instruction, accepted keyboard responses on the second day, specified sophisticated answer analysis on the third day, accepted light pen responses on the fourth day, and began coding their own logic programs on the fifth day. During the first five days the VAULT group was provided with the preprogrammed logic illustrated in Figure 2 (p. 154).

The first item in the HRRC questionnaire was intended to provide an indication as to whether teachers in the workshop found COURSEWRITER much easier, easier, as easy or as difficult, more difficult, or much more difficult to learn than VAULT. Questionnaire respones, illustrated in Table 1 (p. 161), show that nine teachers (75%) found COURSEWRITER more difficult to learn.

Extent of Separation of Logic and Data

VAULT is somewhat similar to PLATO in that the teacher selects a logic and provides the subject matter, answer processing rules, etc. Usually, the VAULT author enters a single program which contains both the logical structure of the course and the corresponding subject matter.

It is possible to compile VAULT logic material without any accompanying data, but VAULT does not accept any data unless data material has been immediately preceeded by material from the logic division. Once a logical structure has been programmed, for a specific course, it is possible to supply various types of subject matter to compliment the logic program, providing the data conforms to the logic. This does not mean, however, that a teacher-author is restricted only to available programmed logics. It is possible to reverse the procedure and have a logic program written to suit specific data formats.

Teacher-authors who only occasionally program materials would probably rely upon available programmed logics. Thus, once a logic program has been tested and researched by educational researchers it would be made available to teachers.

During the HRRC workshop, both groups of teachers received training in VAULT and each teacher was provided with a common pre-programmed, documented logic program. An illustration of this pre-programmed logic is contained in Figure 2 (p. 154). The teachers' attitudes toward the provided logic program were measured by items four and five of the questionnaire. Each teacher responded to the two items in terms of five categories of agreement or disagreement.

Table 4, which represents teacher responses to item four, shows that seven teachers (59%) agreed and one teacher (9%) strongly agreed that the logic program, which was supplied, restricted the manner in which they could present course material. Teacher reaction to the pre-programmed logic possibly could have arisen due to the fact that the provided logic program was primarily intended as a teaching aid. As they progressed through each LESSON of the logic program the teachers easily learned to match data DCR's to those in logic as well as learned which logic VERBS required conformable keywords in data.

Response to questionnaire item five is contained in Table 5. The results from Table 5 indicate nine teachers (77%) agreed and two teachers (17%) strongly agreed that if they were to write more VAULT programs they would like to write their own logic material. This reaction, of course, could be a carry over from item four of the questionnaire. However, it is likely that the teachers felt they could adequately program logics to suit their own individual needs. It must be remembered that at the end of the two week training period VAULT authors were providing their own logic and data material. Another possible reason for teachers wanting to write their logic programs was the fact that the entire group represented a varied range of subject matter specialization. Because of their varied interest areas, different logic programs would have been needed to handle the types of presentations they wished to make. An interesting point is that, in most cases, the teachers were able to modify their data to conform to the original logic program. At present, there appears to be a need for a wide range of logics to be made available for use by teachers who have mastered the rudiments of entering VAULT data material.

Ease of Handling Subject Matter

VAULT provides the teacher-author with a quick and easy method of entering CBI material. Separation of course logic from course content is advantageous to the teacher using VAULT because now he can concentrate upon the task of entering subject matter to match a specific course logic. This means that the teacher needs only to learn enough data division procedures to be able to handle simple logic programs. As the teacher learns more VAULT data procedures the easier it will be for him to move on to use more complicated logic programs. Since VAULT provides the logic author with system default options and since specification of parameters is easily learned, any teacher who requires a specific course logic could quickly learn to write his own logic programs.

The data author who initially uses VAULT would find that the most obvious labor saving feature provided is the editing routine. The teacher merely enters textual material and VAULT formats the text according to the paramters specified by the corresponding VERB in the logic division. If a particular word exceeds the space available at the end of a CRT line that word would be checked to determine whether it is to be hyphenated. Any word, containing less characters than the minimum required for hyphenation, would be displayed on the next CRT line. Control over the size of words to be hyphenated is maintained by the data author although he could use the system hyphenation default.

The logic author is provided with a number of VAULT system defaults designed to simplify entry of logic verb parameters. These system defaults could be overridden by having the author enter specific parameters. Further, VAULT provides the logic author with a small measure of mathematical ability through the use of the verb ASSIGN. However, since the final product is COURSEWRITER source code, there is, at present, no mathematical capability passed on to the student.

All entry of VAULT material is accomplished through off-line procedures. Logic and data entry material, in the form of punched cards, is run in batch mode on the System 360/67. Final COURSEWRITER source generated by VAULT is stored either on tape or punched cards. Finally, the VAULT produced COURSEWRITER source is assembled on the 1500 System. Contained in Appendix F is a copy of a sample VAULT program.

A number of experienced COURSEWRITER authors who have used VAULT expressed concern regarding the off-line authoring procedure. Most of the experienced COURSEWRITER authors at the University of Alberta had previously entered their programs on-line and therefore were apprehensive about the off-line turn around time of one day. This means that, once a VAULT program is entered for processing, there is no immediate feedback regarding error free compilation of the program as there would be during on-line COURSEWRITER authoring.

However, the teachers in the HRRC workshop did not seem to share the concern of the experienced authors regarding on-line authoring. Actually, the only on-line procedures conducted by the VAULT authors was execution of their programs. During on-line sessions, authors executed their COURSWRITER programs and checked the presentations against the data entry material which was illustrated in their VAULT DATA listings. Program changes, if any, were usually noted on the VAULT DATA listings. Rather than making changes on-line, teachers were encouraged to edit their VAULT entry cards. A special set of characters ("¬#") enabled teachers to modify only those data records containing errors. Once the VAULT cards were edited the teachers would resubmit their programs for processing on the System 360/67.

A teacher who wishes to redefine a particular CRT display area would need only to change the particular logic DISPLAY parameter(s) in question and resubmit the entire VAULT program. Thus a teacher-author could make minor modifications to either logic or data entry material and VAULT would reformat and edit the material to desired specifications. This means that a teacher could easily make changes in a program without tediously recoding the material as would be required with COURSEWRITER.

During their on-line sessions the VAULT group, from the HRRC workshop, concentrated mainly upon debugging their programs through the use of the execution control mode. They would execute entire programs or sections of programs. Each textual display, for example, would be checked for spelling errors, ease of reading, placement of text, etc. Upon encountering a response request (QUESTION), the teachers would enter every expected answer as well as a few unexpected answers to check the answer matching routines, accompanying reinforcement messages, and the resulting system action.

It is interesting to note that during on-line sessions VAULT authors preferred not to execute programs while most terminals were being used by COURSEWRITER authors for on-line entry of course material. These teachers indicated that system reaction time was slowed considerably when a number of authors simultaneously used the 1500 System for purposes of on-line program entry.

The third item of the HRRC questionnaire requested teachers from the workshop to compare the estimated amount of authoring time required to write a program using VAULT and the time required to write a similar program using COURSEWRITER. Results of teacher responses to item three are illustrated in Table 3 (p. 162). Ten teachers (85%) estimated that a program written in VAULT took much less authoring time than a like program in COURSEWRITER.

Examination of Table 8 (p. 164) discloses that teachers who initially received instruction in VAULT produced 18.7 times as many lines of VAULT generated COURSEWRITER code, per hour of authoring time, than they did lines of COURSEWRITER during the COURSEWRITER instructional period. The second group, which initially received training in COURSEWRITER, produced 11.6 times as many lines of VAULT generated COURSEWRITER, per hour of authoring time, during the VAULT training session, than lines of COURSEWRITER material produced per hour of authoring during the initial COURSEWRITER session. Overall, the ratio was 13.6 times as many lines of COURSEWRITER produced, per hour of authoring, using VAULT rather than COURSEWRITER.

The results from the HRRC workshop indicate that VAULT effectively increased the number of lines of COURSEWRITER code produced per hour of authoring. One teacher commented, in section G of the questionnaire, that programming in VAULT gave him a psychological advantage over programming similar material in COURSEWRITER.

A number of aspects related to entry of VAULT material, however, appear to be in need of revision or improvement. For example, card columns 72 through 80 of each input record are ignored; perhaps this portion of the card could be used for data entry and thus reduce some confusion that results from use of various types of coding forms. Also, some method of providing numeric capability, for use by students is urgently needed. Provision of the verb ASSIGN was a valuable step in the direction of allowing mathematical variables but more mathematical power is needed. Perhaps a verb (i.e., INCREMENT) could be used to enable a particular variable to be incremented by a particular numeric value, either negative or positive (i.e., INCREMENT X by 2; Y by -1). Thus each time the variables X and Y would be encountered, in logic, they would be incremented by the value specified in the "INCREMENT" record. If an author wishes to change the value by which a variable is to be incremented he would need only to enter a new INCREMENT record with different parameter specifications.

Ease and Power of Response Analysis

The method whereby a teacher-author supplies data answer lists and reinforcement messages is very simple to learn and easy to use in VAULT. However, VAULT does not provide the author with a very powerful type of analysis for evaluating students' keyboard responses. As mentioned previously, VAULT generated COURSEWRITER source code is assembled on the 1500 System. This signifies that if sophisticated answer analysis is to be performed the program source code must contain statements which call special COURSEWRITER functions. The VAULT subset, unfortunately, does not automatically call any COURSEWRITER functions. This single feature restricts, to a large extent, the type of analysis and editing of student responses that may be achieved through use of VAULT.

To overcome this current drawback a logic author could activate specific COURSEWRITER functions, to edit or analyze student responses, by inserting appropriate COURSEWRITER code between the logic DCR's $$CWSTART and $$CWEND and placing these records in the desired location in logic. The disadvantage of using this method for calling COURSEWRITER functions is that the logic coding could eventually contain a great deal of COURSEWRITER statements. If this happens, then, in some instances, an author would be further ahead to code the entire program in COURSEWRITER. Also, if COURSEWRITER code is used in portions of logic, a further restriction is imposed because the logic author must then have an adequate grasp of COURSEWRITER.

The lack of power in the analysis of student supplied answers mainly concerns the need for a few improvements in versatility and power of the VAULT logic division. Provision of a VAULT system default to edit each keyboard response, more specifically to downshift a response, would provide automatic inclusion of the COURSEWRITER "EDIT" function and also decrease coding effort associated with specification of teacher supplied answer lists. For example, if the teacher expects the answer "RED" to be entered in response to a question then his supplied answer list would likely contain three versions of the answer (i.e., CORRECT: RED;<RED;<R>ED□□). If the edit default was included in VAULT only one answer would be required (i.e., CORRECT: RED□□). This additional feature could save coding time especially if a large number of expected answers are to be listed by the teacher-author. As is the case with the other system defaults a teacher should have the option to override the edit default.

A positive feature provided by VAULT logic is the default conditional branching associated with a student's performance within any problem. Whenever the system automatically branches a student back a number of problems, due to system provided defaults or author specified defaults provided by logic minor DCR's, a random message is displayed on the CRT notifying the student he will be reviewing previous material. One possible addition to this method of supplying default branching would be to allow a logic author to enter parameters with the logic minor DCR to allow either backward or forward default branching (i.e., $$WRONGS YES,3,-1 or $$WRONGS YES,3,2).

It is noteworthy that system default branch statements could be preceeded by conditional branches entered through use of the logic IF---THEN records. However, complex conditional branch statements, such as those possible in PLANIT, can not be used in VAULT logic. Only a single condition is permitted in each VAULT IF record.

The writer noted a particular weakness in VAULT IF--THEN statements and their versatility in response analysis. Each THEN record may only be followed by associated GO TO action. There are occasions, though, when the author may desire a number of system activities to be performed when a certain condition is satisfied. For example, if a conditional statement is satisfied the author may wish to initiate the following series of activities: erase a portion of the CRT, display a message, open the image projector, pause for a designated period of time, close the image projector, and branch to a new lesson. It is possible to perform all the above activities within the scope of the current VAULT subset but with considerably more code than the following:

```
IF CORRECT THEN PERFORM
ERASE X,Y
DISPLAY INSERT A,B,C,D
SHOW OPENED,456,TIME6
GO TO NEXT1
END PERFORM
```

The VAULT subset seems to provide teacher-authors with a much easier method of entering answers than does COURSEWRITER. A certain lack of power is evident in the VAULT logic division although all desired types of response analysis and conditional execution of particular groups of statements could be accomplished through the current VAULT subset. The addition of certain improvements to the logic division such as the edit default option, specification of COURSEWRITER functions, do-loop action, and complex IF (condition) statements would increase the ease and power of response analysis.

Maintenance of and Access to Student Accounting Information

A time consuming portion of authoring, that of maintaining student accounting information, appears to have been substantially simplified and reduced by VAULT. System default options, available to logic authors, provide a means for maintaining student accounting information through automatic updating of specific counters, switches, buffers, and return registers. Thus, once a logic program is prepared, the teacher-author is not burdened with the task of entering code to continually update performance data.

The logic minor DCR's (i.e., $$CORRECTS) enable the logic author to specify various accounting defaults to be used within particular BLOCK, LESSON, or PROBLEM levels. Thus a particular number of counters used for student accounting, by VAULT, can be activated or disabled for specific levels of logic by use of the parameter "NO" on the appropriate logic minor DCR. The counters used for accounting purposes provide information as to the cumulative number of PROBLEMS presented, attempts made, and answers which were timed out, correct, and unrecognizable or wrong within the course the current BLOCK, and the current LESSON. As well, other counters at the current PROBLEM level maintain the time taken to respond to a question, number of timeouts, unrecognizables, wrongs, and the numeric value representing the position of the student's answer within the teacher supplied list of additional answers. Therefore, by simply using the system default options an author's task could be greatly simplified and, of course, made easier. A drawback which exists, even with the added accounting facilities provided by the VAULT subset, is that no provision exists for logic author to use VERBS such as ADD, MULTIPLY, LOAD, etc., to perform operations upon the accounting devices.

Another highly useful feature of VAULT is the provision of a unique response identifier with each enter and process statement. This means that performance recordings are much simpler to analyze since each response is always identified by a ten character identifier. The identifier denotes the BLOCK number, LESSON number, and PROBLEM number where the response originated in data as well as the label associated with the corresponding QUESTION or POINT in logic. In addition, each teacher supplied answer within a PROBLEM is associated with a unique two character identifier.

The final noteworthy student accounting aspect of the VAULT subset is provision of restart points. When a student signs off a course and then returns to continue the course, at some later period of time, he begins from the last restart point encountered during the previous session. This does not mean that he necessarily begins from the problem last executed. Therefore, within each LESSON, VAULT provides restart points at the first PROBLEM and at every tenth PROBLEM in that lesson. This means that a student who returns to continue a course could repeat a maximum of only nine PROBLEMS.

The VAULT procedures described above represent a step forward in maintaining student accounting information and providing access to that information. However, these procedures do not fully meet the ideal situation where performance data, from any portion of the course, could be directly accessed during program execution time.

Control Over Various Types of Interface Devices

The data author, in entering subject matter, does not need to worry about entering instructions to control interface devices. Control over interface devices would have been previously defined by the logic author. Actually, the VAULT logic author has control over all interface devices except the audio unit and display of graphic characters. These two VAULT omissions could be activated, however, through use of the logic DCR's $$CWSTART and $$CWEND.

Meaningful logic VERBS, such as DISPLAY or SHOW, enable the author to display textual material, to accept keybard or light pen responses, and to activate the image projector for varying periods of time. The teacher-author who uses a preprogrammed logic merely supplies the needed textual material or the new film frame number if the number specified in the logic program differs from the frame number of the film to be shown. Further, VAULT permits the teacher to change dictionaries where needed in the display of textual material.

Parallelism With Existing COURSEWRITER

In order to provide a comprehensive evaluation of the VAULT subset this writer selected two diverse programs, already available in COURSEWRITER, at the University of Alberta, and prepared two corresponding VAULT courses that were parallel in logic and data to the two COURSEWRITER courses. Emphasis was placed upon discovery of particular types of COURSEWRITER produced action that could not be duplicated by the

VAULT subset.

One program, a patient-management simulation, intended for third and fourth year medical students, takes subjects through a number of various paths that could be followed by a medical doctor in consultation with a fictious patient. Stages of the interview included obtaining a medical history, performing a physical examination, requesting and receiving laboratory tests, diagnosis and therapy. The second program, on the other hand, was for a graduate course in educational psychology (analysis of variance) that mainly provided information in a linear fashion but permitted the student a great deal of control in movement through the course.

The results were very encouraging in that both programs were quite easily duplicated, by the VAULT subset, thus indicating that VAULT could be used to produce a wide range of programs already available in COURSEWRITER. In addition, this writer found that with slight modification a simple basic logic could have been produced by VAULT. This means that new VAULT programs could be produced quite easily merely by providing different subject matter, but the reverse would be difficult to achieve in COURSEWRITER.

The above procedure is not recommended for conversion of existing COURSEWRITER programs to VAULT logic and VAULT data. Ideally, the precise documentation of the COURSEWRITER logic should be used to specify VAULT logic since varying amounts of inefficiency are introduced in abstracting COURSEWRITER logic from a COURSEWRITER program and the procedure does not guarantee the optimum definition of the VAULT logic division.

CHAPTER VII

SUMMARY AND IMPLICATIONS

Originally this study began due to two major drawbacks encountered by teacher-authors at the University of Alberta, who were using COURSEWRITER as an authoring language in producing computer based instructional material. Firstly, the teachers found COURSEWRITER a difficult language to learn and, secondly, an excessive amount of authoring time was required to produce one hour of student terminal time. Therefore, alternative methods of authoring materials for the 1500 System were investigated and the final result was a new teacher oriented authoring language called VAULT.

The study consisted of the following five phases: investigation, design, development, implementation, and evaluation. In the initial portion of the investigative phase a review of literature related to CBI languages was conducted. Findings from the literature review were combined with the expressed needs of CBI teacher-authors at the U of A and resulted in a set of criteria to be used in evaluation CBI languages. A number of current CBI languages, including COURSEWRITER, were then evaluated. An outcome of this evaluation was a list of CBI language features deemed desirable for inclusion in an authoring language. Consideration was given to augmenting COURSEWRITER to include these features. However, due to anticipated difficulties in rewriting the COURSEWRITER assembler and portions of the 1500 System a decision was made to design a completely new high-level authoring language.

The second phase consisted of designing the language. Further examination of the criteria indicated that VAULT should be divided into two divisions (LOGIC and DATA). The logic division was intended for use by persons skilled in CBI. This division accepts high-level statements that set the instructional strategy for the course while data division, on the other hand, provides the teacher with an easy method for entering subject matter. A restriction is that VAULT data must conform, in certain respects, to the framework set down by the logic program.

In order that authors could easily enter material to both divisions natural, educationally oriented terms were used. An input record, called the data control record (DCR), was used in each division to subdivide a course into logical groupings. The logic DCR is more sophisticated than the data DCR because it offers an author some control over a series of system defaults. System defaults were provided to automatically initiate maintenance of student performance data and various branching procedures and thereby reduce authoring time associated with these procedures as is the case in COURSEWRITER.

A series of meaningful action words, called VERBS, were made available to the logic author. VERBS specify the type of subject matter expected from the data division, specific computer units to be used and, also, control the logic flow of the program during student execution time. A set of paramters is usually associated with each VERB. Parameters would usually be entered to regulate the computer unit or indicate conditional course flow. Default parameters are supplied to most VERBS, by VAULT, so that beginning logic authors could quickly learn to program "simple logics" without necessarily knowing that VERBS contain associated parameters.

The data division permits the teacher to easily enter course material by means of KEYWORD records. Most KEYWORDS chosen are words that often are used by teachers in classroom situations (i.e., QUESTION, CORRECT, WRONG, etc.).

Since portions of the data entry material must conform (synchronize) with the logic program KEYWORDS were classified into two forms, conformable and non-conformable. A teacher would sequentially follow the logic statements, within a particular level (domain), and when a VERB that requires data material is encountered he would supply matter on an appropriate KEYWORD record. The non-conformable KEYWORD, on the other hand, permits the teacher to supply answer lists and reinforcement messages. Although the logic program controls the type of data entry records to be input an author does have the option to control movement through the various logic domains.

Some of the features initiated by VAULT logic were: (a) system default options for automatic maintenance of student accounting information at various logic levels; (b) system default options for specifying conditional branching based upon the student's previous performance; (c) provision of some mathematical ability to the author through use of the compile time verb ASSIGN; and (d) default parameters for a number of VERBS. Prominent features provided by the data division were: (a) editing of textual display material; (b) some control over the hyphenation (breaking) of words; (c) editing of strings of answers entered in answer lists; (d) use of display labels for identifying CRT areas in specification of light pen answers; (e) randomly selected reinforcement messages; (f) unique response and answer analysis identifiers; and (g) the option to add material to a previously constructed course.

Once the basic model of VAULT was designed the language was ready to be developed. Due to a number of difficulties anticipated in using the IBM 1130 System, the IBM System 360/67 was chosen as the computing system on which VAULT was to be implemented. A VAULT compiler, that could be described as table driven, was coded in PL/1 and implemented on the System 360. The function of the compiler was to process logic material and then integrate output from the logic division with data material supplied to the data division. The final output, COURSEWRITER source code, is then available to be assembled on the 1500 System.

In July, 1969, a group of twelve teachers attended a four week workshop for training in preparing computer based instructional material. The teachers were randomly divided into two groups of six. During the initial two weeks one group received training in COURSEWRITER and the other group received training in VAULT. During the last two weeks the groups were reversed and received training in the other language. This workshop represented the first major attempt at having VAULT implemented by teachers.

Following the workshop a questionnaire (Appendix A) was completed by each teacher. Results of the questionnaire indicated that the amount of authoring time required to produce CBI material in VAULT was reduced by about a factor of 13 when compared with similar material authored in COURSEWRITER. Also, the teachers indicated that COURSEWRITER was more difficult to learn that VAULT. It is noteworthy that all 12 teachers expressed interest in using the 1500 System during the following year.

There were very few "bugs" in the VAULT compiler throughout the workshop period and in the ensuing period since the workshop. The only major problem that arose was the lack of a logic manual which, at the time, was just being written. Originally the teachers had difficulty in understanding the correspondence between logic and data. This prompted the writting of an author's concepts and facilities manual. Other areas of VAULT that caused some difficulty were: (*a*) the lack of a symbol for specifying that textual material is to be displayed on a new screen; (*b*) the lack of a symbol to improve the ease with which teachers could edit data records containing keypunching errors; (*c*) the need for an improved hyphenation routine; and (*d*) the format of listings of the VAULT logic entry records and data entry records was to compact making it difficult for authors to easily find the PROBLEM levels. As each of the above areas of difficulty were encountered, by the teachers, they were improved upon during the workshop.

Although only a subset of VAULT was developed an experienced VAULT author could produce most programs that can currently be written in COURSEWRITER. However, the process could become quite complicated. Therefore, there is need for a number of major improvements to the VAULT subset. These improvements could be achieved by applying a majority of the features from the original design specifications of VAULT.

Following are some suggested improvements to the VAULT logic division. Firstly, in conjunction with the logic minor DCR's, an author should be provided with capabilities for positive or negative default branching. Further, the default conditions should automatically include the COURSEWRITER "edit," "keyletter," and "order" functions, but with provision that a logic author could override these defaults. In addition a number of minor DCR's should be made available to allow an author to completely suppress system defaults which automatically alter the contents of counters, switches, return registers, etc. (i.e., $$COUNTERS).

The types of logic statements should also be expanded to permit new VERBS, more powerful IF statements, and another compile time VERB. A few of the VERBS which should be added: (*a*) PERFORM--to permit do loop action; (*b*) LOAD, ADD, SUBTRACT, MULTIPLY, and DIVIDE--to allow expanded scorekeeping instructions; (*c*) CALL--to activate COURSEWRITER functions; (*d*) PLAY and RECORD--to permit use of audio units; and (*e*) RETRY--to specify that the student is to attempt another response to the question. In addition the word GRAPHIC would be available as an optional parameter to accompany the verb DISPLAY.

Currently, the THEN statement only allows the verbs GO TO and STOP to be included as action words. Expansion of the VERBS permitted as action words in the THEN statements would permit an author to enter statements such as:

```
IF WRONG THEN PERFORM
  DISPLAY
  PAUSE 100
  PLAY
  SHOW OPENED
  RETRY
END PERFORM
IF CORRECT
THEN LOAD COUNTER-15 INTO COUNTER-18
```

Addition of the compile time verb INCREMENT would increase the ease with which variables could be used. For example, the statement "INCREMENT X by 2" would cause the variable X to be incremented by the value 2 each time X is encountered by the compiler.

Basically the data division requires very few additions. The single major improvement that is needed is a feature which permits VAULT to recover from situations where data entry material is out of synchronization with the logic output material. One possibility, when a synchronization error occurs, it to accept and list data input records up to the next data LESSON but not to perform any syntax checking. At the next data LESSON, however, VAULT would move on to the next LESSON in logic and resume processing data records. Another addition would be to modify the text editing routine to include right justification of textual material to appear on the CRT.

There are a number of implications concerning the use of VAULT in the area of computer based instruction. By separating the logic and data portions of a course VAULT provides education with a vehicle whereby teachers could become activly involved in the preparation of CBI materials. As more logic programs are tested, researched, and made available then more varied types of courses can be made available. Also, if VAULT takes less time to learn and also, less time to author course materials costs associated with production of CBI materials would also be reduced. This would be an important economic factor in the decision of school districts in accepting costs associated with computer based instruction.

Further, VAULT holds certain implications to educational researchers. Now it is possible to define instructional strategies and program these through use of the VAULT logic division. Each logic could then be tested by using various types of subject matter to compliment these logics. In addition, it is possible to write various logic strategies and apply them to a relatively fixed set of subject matter.

SELECTED REFERENCES

Atkinson, R. C. Instruction in initial reading under computer control: The Stanford Project. Journal of Educational Data Processing, 1967, 4, 175-192.

Berlyne, D. E. Conflict arousal and curiosity. New York: McGraw Hill, 1960.

Bernstein, R. RCA Instructional Systems Conference, 1967, 1. Cited by J. L. Rogers, Current problems in CAI. Datamation, 14(9), 1968. P. 32.

Bitzer, D. L. Some pedagogical and Engineering design aspects of computer-based education. Paper presented at the 16th International Congress of Applied Psychology, 1968.

Bitzer, D. L., Braunfeld, P. G., & Lichtenberger, W. W. PLATO II: A multiple-student computer-controlled automatic teaching device. In J. E. Coulson (Ed.), Programmed learning and computer based instruction. New York: Wiley, 1962. Pp. 205-216.

Bitzer, D. L., Lyman, E. R., & Easley, J. A. Jr. The uses of PLATO: A computer-controlled teaching system. Report R-268, Urbana: Coordinated Science Laboratory, University of Illinois, 1965.

Bitzer, D. L., Hicks, B. L., Johnson, R. L., & Lyman, E. R. The PLATO system: Current research and developments. IEEE Transactions on Human Factors in Electronics, 1967, HFE-8(2), 64-70.

Bolt Beranek and Newman Inc. TELCOMP Manual. Cambridge, Mass: TELCOMP Services, 1966.

Borko, H. (Ed.) Computer applications in the behavioral sciences. Englewood Cliffs, N. J.: Prentice-Hall, 1962.

Briggs, L. J. Two self-instructional devices. Psychological Reports, 1958, 4, 671-676.

Bushnell, D. D. Applications of computer technology to the improvement of learning. In D. D. Bushnell and D. W. Allen (Eds.), The computer in American education. New York: Wiley, 1967. Pp. 59-76.

Bushnell, D. D., & Allen, D. W. The computer in American education. New York: Wiley, 1967.

Chapman, R. L., and Carpenter, J. T. Computer techniques in instruction. In J. E. Coulson (Ed.), Programmed Learning and computer-based instruction. New York: Wiley, 1962. Pp. 240-253.

Coulson, J. E. (Ed.) Programmed Learning and computer based instruction. New York: Wiley, 1962.

Coulson, J. E. Automation, electronic computers, and education. *Phi Delta Kappan*, 1966, 47, 340-344.

Coulson, J. E., & Silberman, H. F. *Results of initial experiment in automated teaching*. Santa Monica: System Development Corporation, July, 1959.

Coulson, J. E., & Silberman, H. F. Effects of three variables in a teaching machine. *Journal of Educational Psychology*, 1960, 51, 135-143.

Crowder, N. A. Automatic tutoring by means of intrinsic programming. In E. H. Galanter (Ed.), *Automatic teaching: the state of the art*. New York: Wiley, 1959, Pp. 109-116.

Dick, W. The development and current status of computer-based instruction. *American Educational Research Journal*, 1965, 2, 41-51.

Ellis, T. O., & Sibley, W. L. On the development of equitable graphic I/O. *IEEE Transactions on Human Factors in Electronics*, 1967, HFE-8(1), 15-17.

Entelek box score on CAI programs. Newburyport, Mass.: Entelek Inc., May, 1967.

Falkoff, A. D., & Iverson, K. E. *APL/360: User's Manual*. Yorktown Heights, New York: IBM Corporation, 1968.

Fano, R. M., & Corbato, F. J. Time sharing on computers. In D. Flanagan (Ed.), *Information*. San Francisco: Freeman and Co., 1966. Pp. 76-95.

Feingold, S. L. PLANIT (Programming language for interactive teaching). *Third ONR Conference on CAI*, Santa Monica: System Development Corporation, 1966, September, 4-6.

Feingold, S. L. PLANIT - A language for CAI. *Datamation*, 1968, 14(9), 41-47.

Feingold, S. L., & Frye, C. H. *User's Guide to PLANIT: Programming language for interactive teaching*. Technical Memorandum TM-3055/000/01, Santa Monica: System Development Corporation, October, 1966.

Feurzeig, W. A. A conversational teaching machine. *Datamation*, 1964, 10(6), 38-42.

Feurzeig, W. A. New instructional potentials of information technology. *IEEE Transactions on Human Factors in Electronics*, 1967, HFE-8(2), 84-88.

Frye, C. H. CAI languages: Capabilities and applications. *Datamation*, 1968, 14(9), 34-37.

General Electric Co., *BASIC language*. (Rev. ed.) Bethesda, Maryland: Information Service Department, 1967. (a)

General Electric Co., *Introduction to programming in BASIC*. (Rev. ed.) Bethesda, Maryland: GE Information Service Department, 1967. (b)

Gentile, J. R. *The first generation of computer-assisted instructional systems: An evaluative review*. University Park, Pa.: Computer Assisted Instruction Laboratory, Pennsylvania State University, November, 1965.

Gerard, R. W. Computers: Their impact on society. *AFIPS Conference Proceedings*, 1965, 27(2), 11-16.

Gerard, R. W. (Ed.) *Computers and education*. New York: McGraw-Hill, 1967.

Glaser, R. Some research problems in automated instruction: Instructional programming and subject-matter structure. In J. E. Coulson (Ed.), *Programmed learning and computer-based instruction*. New York: Wiley, 1962. Pp. 67-85.

Glaser, R., Homme, L. E., & Evans, J. L. An evaluation of textbooks in terms of learning principles. A paper read at the meeting of the American Educational Research Assoc., Atlantic City, N. J., Feb., 1959. In A. A. Lumsdaine and R. Glaser (Eds.), *Teaching machines and programmed learning: A source book*. Washington: National Education Association, 1960. Pp. 437-445.

Glaser, R., Ramage, W. W., & Lipson, J. I. *The interface between student and subject matter*. Pittsburg: Learning Research and Development Center, University of Pittsburg, 1964.

Gleason, G. T. Computer assisted instruction - prospects and problems. *Education Digest*, 1968, 33, 14-17.

Goldstein, L. S., & Gotkin, L. G. Review of research: Machine vs. text. *Journal of Programmed Instruction*, 1962, 1, 29-36.

Hansen, D. N. Computer assistance with the educational process. *Review of Educational Research*, 1966, 36, 588-603.

Hickey, A. D., & Newton, J. M. *Computer-assisted instruction: A survey of the literature*. (2nd ed.) Newburyport, Mass.: Entelek Inc., January, 1967.

Hunka, S. M. Introduction to APL 360/67 programming. *Research and information report CAI-5-67*. Edmonton: Educational Research Services, University of Alberta, 1967.

International Business Machines Corporation. *IBM 1500 Coursewriter author's guide Part I: Course planning*. San Jose, Calif.: IBM Corporation, 1967. (a)

International Business Machines Corporation. IBM 1500 Coursewriter II author's guide Part II: Course program development. San Jose, California: IBM Corporation, 1967. (b)

Iverson, K. E. A programming language. New York: Wiley, 1962.

Lee, J. A. The anatomy of a compiler. New York: Reinhold Publishing, 1967.

Licklider, J. C. R. Preliminary experiments in computer-aided teaching. In J. E. Coulson (Ed.), Programmed learning and computer-based instruction. New York: Wiley, 1962. Pp. 217-239.

Lumsdaine, A. A. Teaching machines and self-instructional materials. Audio-Visual Communication Review, 1959, 7, 163-181.

Lumsdaine, A. A., & Glaser, R. (Eds.) Teaching machines and programmed learning: A source book. Washington: National Education Association, 1960.

Lyman, E. R. Instructions for using the PLATO logic, GENERAL. CERL Report X-1, Urbana: Computer-based Education Research Laboratory, University of Illinois, May, 1968.

Maher, A. Computer-based instruction (CBI): Introduction to the IBM project. Research Report RC1114, White Plains, N. Y.: IBM, 1964.

Mitzel, H. E. (Ed.) The development and presentation of four college courses by computer teleprocessing. University Part, Pa.: United States Department of Health, Education, and Welfare, June, 1967.

Myer, R. H. TELCOMP manual for users. Cambridge, Mass.: Bolt Beranek and Newman Inc., 1966.

Pressey, S. L. A simple apparatus which gives tests and scores - and teaches. In A. A. Lumsdaine, and R Glaser (Eds.), Teaching Machines and programmed learning: A source Book. Washington: National Education Association, 1960. Pp. 35-41. Reprinted from School and Society, 23, 1926.

Pressey, S. L. A machine for automatic teaching of drill material. In A. A. Lumsdaine and R. Glaser (Eds.), Teaching machines and programmed learning: A source book. Washington: National Education Association, 1960. Pp. 42-46. Reprinted from School and Society, 25, 1927.

Pressey, S. L. A third and fourth contribution toward the coming "industrial revolution" in education. In A. A. Lumsdaine and R. Glaser (Eds.), Teaching machines and programmed learning: A source book. Washington: National Education Association, 1960. Pp. 47-51. Reprinted from School and Society, 36, 1932.

Ragsdale, R. The Learning Research and Development Center's computer assisted laboratory. *DECUS Proceedings Fall 1965*, Maynard, Mass.: Digital Equipment User's Society, 1966, 65-68. Cited by D. N. Hansen, *Review of Educational Research*, 1966, 36, 590.

Rath, G. J., Anderson, N. S., & Brainerd, R. C. The IBM Research Center teaching machine project. In E. H. Galanter (Ed.), *Automatic teaching: the state of the art*. New York: Wiley, 1959. Pp. 117-130.

Rodgers, W. A., & Gariglio, L. M. *Toward a computer based instruction system*. Saginaw, Michigan: Saginaw Township Community Schools, 1967.

Rogers, J. L. Current problems in CAI. *Datamation*, 1968, 14(9), 28-33.

Schramm, W. *The research on programmed instruction: An annotated bibliography*. Washington: United State Department of Health, Education, and Welfare, 1964.

Silberman, H. F. Overview of CAI at SDC. *Third ONR Conference on CAI*. Santa Monica: System Development Corporation, 1966, September, 1-2.

Silberman H. F., & Coulson, J. E. Automated teaching. In H. Borko (Ed.), *Computer applications in the behavioral sciences*. (2nd ed.) Englewood Cliffs, N. J.: Prentice Hall, 1962. Pp. 308-335.

Silberman, H. F., & Rosenbaum, J. Computer-based instruction in statistical inference. *Technical Memorandum TM-2914/003/000*. Santa Monica: System Development Corporation, March, 1967.

Skinner, B. F. The science of learning and the art of teaching. *Harvard Educational Review*, 1954, 24, 86-97.

Skinner, B. F. Teaching machines. *Science*, 1958, 128, 969-977.

Stolurow, S. M. Computer-based instruction: Psychological aspects and systems conception of instruction. *Journal of Educational Data Processing*, 1967, 4, 193-215.

Stolurow, L. M., & Davis, D. Teaching machines and computer-based systems. In R. Glaser (Ed.), *Programmed learning II: Data and direction*. Washington: National Education Association, 1965. Pp. 162-212.

Suppes, P. The uses of computers in education. In D. Flanagan (Ed.), Information: *A scientific American book*. San Francisco: W. H. Freeman & Co., 1966. Pp. 157-174.

Swets, J. A., & Feurzeig, W. Computer-aided instruction. *Science*, 1965, 150, 572-576.

Travers, R. M. V. Research and theory related to audio visual information. Salt Lake City: University of Utah Bureau of Educational Research, 1964. Cited by K. H. Wodtke, Educational Requirements for a student-subject matter interface, 1967. P. 324.

Uttal, W. R. On conversational interaction. In J. E. Coulson (Ed.), Programmed learning and computer-based instruction. New York: Wiley, 1962. Pp. 171-190.

Weaver, W. Recent contributions to the mathematical theory of communication. In C. E. Shannon and W. Weaver (Eds.), The mathematical theory of communication. Urbana: University of Illinois Press, 1963.

Wodtke, K. H. Educational requirements for a student-subject matter interface. In H. E. Mitzel (Ed.), The development and presentation of four college courses by computer teleprocessing. University Park, Pa.: United States Department of Health, Education, and Welfare, June, 1967. Pp. 319-327.

Zinn, K. L. Computer technology for teaching and research on instruction. Review of Educational Research, 1967, 37, 618-634. (a)

Zinn, K. L. Computer assistance for instruction: a review of systems and projects. In D. D. Bushnell and D. W. Allen (Eds.) The computer in American education. New York: Wiley, 1967. Pp. 77-107. (b)

Zinn, K. L. Summary of programming languages and author assistance in computer-based educational systems. Ann Arbor, Michigan: University of Michigan, March, 1967. (Mimeographed draft) (c)

Zinn, K. L. Instructional uses of interactive computer systems. Datamation, 1968, 14(9), 22-27.

APPENDIX A

VAULT LOGIC PROGRAM PROVIDED TO TEACHERS DURING WORKSHOP

```
          $$LOGIC

          $$BLOCK

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

          $$LESSON     INTRO

          $$PROBLEM    PREFCE

          DISPLAY

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

          $$LESSON     ONE

          $$PROBLEM    QUEONE

          QUESTION

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

          $$LESSON     TWO

          $$PROBLEM    QUETWO

          ERASE

          DISPLAY      INSERT     10,0,2,40                A

          DISPLAY      INSERT     14,0,2,40                B

          DISPLAY      INSERT     18,0,2,40                C

          DISPLAY      INSERT     22,0,2,40                D

          DISPLAY      INSERT     26,0,2,40                E

          POINT                   0,0,8,40/28,0,2,40

          $$ENDLOGIC
```

F   Figure 2. VAULT Logic Program Provided To Teachers
During HRRC Worksop

APPENDIX B

HRRC WORKSHOP QUESTIONNAIRE

HRC WORKSHOP QUESTIONNAIRE

July/69

Age ____________ Sex ________________

Years of university education ________________

Years of teaching experience ________________

Grade levels taught ______________________

Number of previous Computing Science courses _______

CAI language first learned in workshop:
(VAULT or CW) ______________________

I. In the following section you will be presented with a number of statements, each containing a blank. Each statement is followed by a number of short phrases. Please select the letter corresponding to the phrase which you think best expresses your feelings toward the completion of the statement. Then, enter the letter onto the blank.

1. After learning both CW and VAULT, I found, in comparing the two languages, that CW was ____________ to learn.

   a. much easier
   b. easier
   c. as easy or as difficult
   d. more difficult
   e. much more difficult

2. If I were to recommend instruction for teachers wishing to learn how to program instructional materials, I would suggest __________.

   a. CW only
   b. VAULT only
   c. CW followed by VAULT
   d. VAULT followed by CW

3. In comparing the amount of time taken to plan and code my CAI programs, I am of the opinion that the programs written in VAULT took ___________ time than the one written in CW.

   a. much less
   b. less
   c. nor more nor no less
   d. more
   e. much more

4. In using VAULT, I felt that the LOGIC which was originally supplied restricted the manner in which I could present my course material.

____

a. strongly agree
b. agree
c. undecided
d. disagree
e. strongly disagree

5. If I were to write more programs in VAULT, I would rather write my own LOGIC. ____________

a. strongly agree
b. agree
c. undecided
d. disagree
e. strongly disagree

6. Once this workshop is completed, I plan ____________.

a. to continue writing programs by using CW only.
b. to continue writing programs by using VAULT DATA only.
c. to continue writing programs by using VAULT (LOGIC and DATA) only.
d. to continue writing programs by using a combination of CW and one or more divisions of VAULT.
e. not to continue writing programs.

7. One of the most difficult concepts required in learning VAULT was the connection between the LOGIC division and the DATA division. ________

a. strongly agree
b. agree
c. undecided
d. disagree
e. strongly disagree

II. During the past four weeks you have been briefly introduced to two CAI languages. We would greatly appreciate your written criticisms (positive and negative) regarding the following areas:

A. Difficulties encountered while learning concepts in:

1. COURSEWRITER

2. VAULT-DATA

3. VAULT LOGIC

4. On-line authoring

B. Materials used during workshop.

1. COURSEWRITER lab problems

2. VAULT lab problems

3. VAULT DATA manual.

C. Method of CAI language presentations (would you like less lecture and more coding and consulting time, etc.?)

D. Class time

1. Would you rather have had classes at times other than 8-10 a.m. during this summer session? If so, at what times?

2. Did you find the one hour alloted for authoring sufficient enough for you to become familiar with the 1500 System?

E. Would you like to author programs and use the system during the coming year?

Yes_________________________ No _________________________

If yes state the days (Monday to Saturday) and times of day preferred.

F. Money is available for bus transportation and substitute teachers. Do you think you will require either next year?

Yes ______________________ No ______________________

G. Please comment about VAULT regarding general design concepts such as DCR's and keywords, as well as anything you would like to see included in the updated version of VAULT.

F. From the program(s) you have written, please estimate:

	CW	CW produced by VAULT
Number of lines		
Hours of coding time		
Minutes of student execution time		

APPENDIX C

RESULTS OF WORKSHOP QUESTIONNAIRE

TABLES 1-10

TABLE 1

RESPONSE OF BOTH GROUPS TO

QUESTION ONE (WHICH LANGUAGE IS EASIER TO LEARN)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
(a) Much easier	0	1	1
(b) Easier	1	0	1
(c) As easy or as difficult	1	0	1
(d) More difficult	1	2	3
(e) Much more difficult	3	3	6

TABLE 2

RESPONSE OF BOTH GROUPS TO QUESTION

TWO (SUGGESTED SEQUENCE OF INSTRUCTION)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
(a) CW only	0	0	0
(b) VAULT only	2	0	2
(c) CW followed by VAULT	1	0	1
(d) VAULT followed by CW	3	6	9

TABLE 3

RESPONSE OF BOTH GROUPS TO QUESTION THREE (TIME TAKEN TO WRITE PROGRAMS IN VAULT RATHER THAN COURSEWRITER)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
(a) Much less	5	5	10
(b) Less	0	1	1
(c) No more nor no less	0	0	0
(d) More	1	0	1
(e) Much more	0	0	0

TABLE 4

RESPONSE OF BOTH GROUPS TO QUESTION FOUR (SUPPLIED LOGIC RESTRICTED COURSE PRESENTATION)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
(a) Strongly agree	1	0	1
(b) Agree	4	3	7
(c) Undecided	0	1	1
(d) Disagree	0	2	2
(e) Strongly disagree	1	0	1

TABLE 5

RESPONSE OF BOTH GROUPS TO QUESTION FIVE (I WOULD RATHER WRITE MY OWN LOGIC)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
(a) Strongly agree	2	0	2
(b) Agree	4	5	9
(c) Undecided	0	1	1
(d) Disagree	0	0	0
(e) Strongly disagree	0	0	0

TABLE 6

RESPONSE OF BOTH GROUPS TO QUESTION SIX (IN WRITING FURTHER PROGRAMS I PLAN TO USE . . .)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
(a) CW only	0	0	0
(b) VAULT data only	0	0	0
(c) VAULT logic and data only	1	1	2
(d) CW & VAULT	5	5	10
(e) Not to author	0	0	0

TABLE 7

RESPONSE OF BOTH GROUPS TO QUESTION SEVEN

(CONNECTION BETWEEN LOGIC AND DATA WAS DIFFICULT TO LEARN)

	GROUP		
	CW-VAULT	VAULT-CW	TOTAL
Yes	6	6	12
No	0	0	0

TABLE 8

MEAN LINES OF COURSEWRITER SOURCE CODE

PRODUCED PER HOUR BY USING EACH LANGUAGE

	LANGUAGE USED		
GROUP	CW	VAULT	RATIO $\frac{\text{VAULT}}{\text{CW}}$
CW-VAULT	16.69	311.76	18.7
VAULT-CW	7.96	92.56	11.6
Total	10.91	148.18	13.6

TABLE 9

AGES OF WORKSHOP TEACHERS

	MEDIAN	MEAN	ST. DEV.
Females	30	34.67	13.61
Males	34	33.78	7.78
CW-VAULT	37	39.33	8.98
VAULT-CW	28	28.67	4.72
Total	34.5	34.00	8.82

TABLE 10

YEARS OF TEACHING EXPERIENCE

	MEDIAN	MEAN	ST. DEV.
Females	11	14.0	14.7
Males	9	9.4	5.6
CW-VAULT	9.5	15.0	8.9
VAULT-CW	7	7.2	6.2
Total	9.5	10.6	8.1

APPENDIX D

VAULT LOGIC DIVISION SPECIFICATIONS

TABLE OF CONTENTS FOR APPENDIX D

Page

I. Syntax

When coding VAULT LOGIC care must be taken to observe syntax rules and to ensure that data has been entered correctly.

For both DCR's and VERBS a parameter list will quite often be available. If a parameter is enclosed in square brackets, i.e., [parameterl], then it may either be included or omitted completely, as the programmer desires. If a set of parameters are enclosed in parentheses, i.e.,

$$\left\{\begin{matrix}\text{parameter1}\\ \text{parameter2}\end{matrix}\right\}$$

then only one of the parameters must be chosen.

For VERBS where CRT screen coordinates are parameters, care must be taken to ensure that the bounds of the screen are not exceeded. The screen has 32 rows, numbered 0 to 31, and 40 columns, numbered 0 to 39. Thus possible coordinates are: x,y, where x is 0 to 31, and y is 0 to 39. One line of text takes up two rows of the screen however. Therefore there are effectively only sixteen text lines available on the CRT screen.

Variable names and labels are also included as part of the parameter list for some DCR's and VERBS. They can be no more than six characters in length. Also, variable names must start with an alphabetic character and have no imbedded special characters. Labels can be alphabetic or numeric. It should be noted that all labels must be unique.

II. Division Control Records

These records are used to:

(1) set up logical subdivisions of BLOCKS, LESSONS, PROBLEMS;

(2) specify the options that are to be in effect for specific PROBLEMS, LESSONS, and BLOCKS;

(3) signify the beginning and the end of the program;

(4) allow actual Coursewriter code to be inserted in the program, and;

(5) specify options desired for the compilation.

Format of DCR's

All DCR's begin with two dollar signs (i.e., $$PROBLEM). Following the DCR, at least one blank space is required after which the desired parameters may be punched. Parameters may begin anywhere following the DCR and a blank, but must end on or before column 71. Card column 72 has been reserved for use by VAULT.

Any information punched in card columns 73-80 will be ignored to allow this area to be used for card sequencing.

Certain DCR's are optional and may be omitted. If this is done then a standard set of options is used by VAULT if required.

The DCR's are divided into two categories: major DCR's and minor DCR's. Major DCR's such as $$BLOCK, $$LESSON and $$PROBLEM, logically subdivide the program and determine the extent of influence of minor DCR's such as $$LATENCY, $$CORRECTS, $$WRONGS, $$UNRECS, $$TIMEOUTS. For example, if a minor DCR is entered after a $$BLOCK and before the first $$LESSON then its options will usually apply throughout the entire

BLOCK. However, the LOGIC programmer may override any option within any LESSON or PROBLEM simply by entering the same minor DCR but with a different parameter list. Similarly, if a minor DCR is entered after a $$LESSON and before the first $$PROBLEM of that LESSON, then its options will hold for every PROBLEM within that LESSON except for those PROBLEMS containing the same minor DCR with different options. The options in a minor DCR entered after a $$PROBLEM will apply only for that particular PROBLEM and will revert back to the options specified for the LESSON as soon as the PROBLEM is completed.

The use of the minor DCR's in conjunction with the major DCR's is very important, therefore, in determining the logical structure of the program and great care must be taken in sequencing the input cards so that the desired results will be obtained. The method of using major and minor DCR's to control the LOGIC will become clearer as each particular DCR is explained and examples are given.

$$LOGIC

This DCR specifies the beginning of the LOGIC DIVISION input and must be the first card. The remainder of the card following this DCR is disregarded and may be used for comments.

$$ENDLOGIC

The DCR specifies that the last LOGIC DIVISION input card has been reached. This record must be the last one since all further input is ignored. As in the DCR $$LOGIC, the portion of the card following $$ENDLOGIC is disregarded and may be used for comments.

Example:

```
$$LOGIC     DRILL AND PRACTICE SEQUENCE
$$BLOCK
$$LESSON
    .
    .
    .
$$ENDLOGIC  END OF DRILL & PRACTICE
```

$$BLOCK

This DCR specifies that all of the following input records are to form a BLOCK, a logically associated subdivision of a course. Only one BLOCK may be compiled at a time. Each BLOCK may contain up to 99 LESSONS.

A BLOCK may be labelled and branches can be made to this label.

Format:

```
$$BLOCK    [label]
```

Example:

```
$$BLOCK  TRIG30
```

Any "minor" DCR's that immediately follow the $$BLOCK (i.e., they precede the $$LESSON and $$PROBLEM), supply modifiers that will hold for the entire BLOCK unless overridden at a LESSON or PROBLEM level.

Example:

```
$$BLOCK
$$LATENCY  400
$$LESSON   ONE
.
.
.
$$LESSON   TWO
$$LATENCY   50
.
.
.
$$LESSON THREE
.
.
.
$$ENDLOGIC
```

In this example the latency throughout the BLOCK will be 40.0 seconds except during LESSON TWO in which case the LATENCY will be 5.0 seconds. LESSON THREE will have a LATENCY of 40.0 seconds.

$$LESSON

This DCR signifies the beginning of a LESSON, a logically associated group of records that are a subdivision of a BLOCK. All input records up to the next $$LESSON will be part of this LESSON.

A BLOCK may contain from 1 to 99 LESSONS, each of which may contain up to 99 PROBLEMS.

LESSONS may be labelled and branches made to the label.

Format:
```
$$LESSON [label]
```

Example:
```
$$LESSON  SINES
```

Any "minor" DCR's that immediately follow the $$LESSON (i.e., they precede the next $$PROBLEM) supply modifiers that will hold for the entire LESSON unless overridden at the PROBLEM level.

Example:
```
$$LESSON
$$LATENCY  405
$$UNRECS   NO
$$PROBLEM  ONE
.
.
.
$$PROBLEM  TWO
$$UNRECS   YES,3,4
.
.
.
$$PROBLEM  THREE
$$LATENCY  300
.
.
.
$$PROBLEM  FOUR
.
.
.
$$LESSON
.
.
.
```

In the above example the LATENCY will be 40.5 seconds in PROBLEMS ONE, TWO, and FOUR. No accumulation of UNRECOGNIZABLE answers will be done in PROBLEMS ONE, THREE, and FOUR. Only PROBLEM THREE will have a LATENCY of 30.0 seconds. Only PROBLEM TWO will have the UNRECOGNIZABLE answers accumulated.

$$PROBLEM

This DCR signifies the beginning of a PROBLEM, a logically associated group of records that are a subdivision of a LESSON. All input records up to the next $$PROBLEM will be part of this PROBLEM.

A LESSON may contain from 1 to 99 PROBLEMS.

A PROBLEM may be labelled and branches made to that label.

```
Format:         $$PROBLEM [label]

Example:        $$PROBLEM ACUTE
                $$LATENCY 400
                .
                .
                .
                $$PROBLEM OBTUSE
                .
                .
                .
```

In this example the latency in PROBLEM ACUTE will be 40.0 seconds, but in PROBLEM OBTUSE the latency will revert back to the latency specified for the LESSON.

$$LATENCY

This record specifies in tenths of seconds the amount of time that will be allowed for a student to make a response to a question. The range allowed is 0 to 9999 (i.e., 0 to 999.9 seconds). If this record is omitted then a standard LATENCY time of 90.0 seconds will be used.

```
Format:         $$LATENCY  tttt
```

where tttt is the latency time in tenths of seconds.

```
Example:        $$LATENCY  800
```

This specifies that students will be allowed a maximum of 80 seconds to answer a question.

$$CORRECTS

This DCR specifies the action to be taken in a Coursewriter program if a correct response is given by a student. It specifies that the programmer wishes to override the current strategies associated with each BLOCK, LESSON and PROBLEM.

The parameters specify:

(1) Whether or not any system action is to be automatically performed.

(2) The absolute number of correct responses necessary to generate system action.

(3) The number of PROBLEMS that will be skipped by the system if a correct response is made.

Format: $$CORRECTS $\left\{\begin{array}{l}\text{NO}\\ \text{[YES,]mm[,nn]}\end{array}\right\}$

where: mm is the absolute number of correct answers required in a LESSON before the student is skipped forward;
nn is the absolute number of PROBLEMS that will be skipped forward when the branch is taken; and
mm and nn can not exceed 99.

If there are not enough PROBLEMS left in the LESSON to accomodate the branch, then the student will be branched to the beginning of the next LESSON.

If this record is not entered at all, then the system will just pass the student on to the next PROBLEM if he answers correctly.

Example 1:

```
$$LESSON
$$CORRECTS  YES, 10,5
$$PROBLEM
```

In this example, if the student gives ten correct answers then he will skip the following five PROBLEMS. If there are not five PROBLEMS left in the LESSON, then he will skip to the beginning of the next LESSON.

Example 2:

```
$$LESSON  MATH
$$PROBLEM ONE
.
.
.
$$PROBLEM TWO
$$CORRECTS NO
.
.
.
$$PROBLEM  THREE
$$LESSON   MATH01
```

In this example, if PROBLEM TWO is answered correctly, the tally of correct answers will not be incremented. However, the tally will be incremented for the remaining problems within LESSON MATH.

Similarily, if $$CORRECTS NO is placed directly after a $$LESSON, then the tally of correct answers will not be incremented for any correct answers given in this LESSON.

NOTE: It is possible to conditionally branch the student to the next LESSON regardless of the number of PROBLEMS remaining in the LESSON. This is done by specifying nn as 99.

Example 3:

```
$$CORRECTS 5,99
```

In this example the student will be sent to the next LESSON when he answers five PROBLEMS correctly.

$$WRONGS

This DCR specifies the action to be taken by the system if a wrong response is given by a student. It specifies that the programmer wishes to override the current strategies associated with each $$BLOCK, $$LESSON, and $$PROBLEM.

Normally on a wrong answer, the system will branch to repeat the last executed QUESTION or POINT. If however, a student answers the same QUESTION incorrectly three times, then the student will be taken back to repeat the last three PROBLEMS. If there are less than three preceding PROBLEMS in the LESSON then the system will start again at the beginning of the LESSON.

The LOGIC programmer can change this strategy by use of the $$WRONGS card.

Format: $$WRONGS $\left\{\begin{array}{l}\texttt{NO}\\ \texttt{[YES,]mm[,nn]}\end{array}\right\}$

where: mm is the number of wrong answers entered for a particular QUESTION that will cause the student to branch back to review;
nn is the number of PROBLEMS that will be skipped when branching back. The system will never branch back past the start of the LESSON; and
mm and nn cannot exceed 99.

If NO is specified then no action will be taken when a wrong answer is given except to keep repeating the QUESTION until another type of answer is entered. Also, the tally of WRONG answers will not be incremented.

Example:

```
$$PROBLEM  ONE
.
.
.
$$PROBLEM  TWO
$$WRONGS   2,1
```

In this example, if the student gives two wrong answers to PROBLEM TWO, he will be branched <u>back</u> to PROBLEM ONE.

NOTE: It is possible to conditionally branch a student to the beginning of a LESSON regardless of the number of PROBLEMS within the LESSON. This is done by specifying nn as 99.

Example: $$WRONGS 10,99

In this example, if a student responds wrongly to any PROBLEM ten times, he will be sent to the beginning of the LESSON for review.

$$UNRECS

This record specifies the system action to be taken in the event that the student gives an unrecognizable (or unanticipated) response. The system default on an unrecognizable response is to repeat the last executed QUESTION or POINT. If three unrecognizable responses are given, then the student will be branched back three PROBLEMS.

However, the programmer may override the system defaults by specifying appropriate parameters in the $$UNRECS card.

```
Format:        $$UNRECS  { NO             }
                         { [YES,]mm[,nn]  }
```

where: mm is the number of unrecognizable answers given for the current QUESTION that will cause the student to branch back to review;
nn is the number of PROBLEMS that will be skipped when branching back to review; and
mm and nn must not exceed 99.

If NO is specified then the tally of UNRECOGNIZABLE answers will not be incremented and the student will be presented with the same PROBLEM continually until he gives a recognizable answer.

Example: $$UNRECS 4,2

In this example, when 4 unrecognizable answers have been given for the same QUESTION then the student will be skipped back 2 PROBLEMS for review.

It is possible to cause the student to be branched conditionally to the beginning of the LESSON. See the Note under $$WRONGS for details.

NOTE: Outside the PROBLEM level VAULT makes no distinction between UNRECOGNIZABLE and WRONG answers. (i.e., only one counter is used to tally both types of responses at the LESSON and BLOCK levels.)

$$TIMEOUTS

This DCR specifies the system action to be taken in the event the student takes too long to respond (i.e., the student times-out).

Normally when a student "times-out," he will be given the current QUESTION again. However, if he times-out on the same PROBLEM three times then he will be branched back three problems for review. If there are less than three preceding PROBLEMS in the LESSON then he will be branched to the beginning of the LESSON. In either case the system will give the student a message telling him he has timed-out. The programmer may again override the system action by the use of this DCR.

```
Format:        $$TIMEOUTS  {NO            }
                           {[YES,]mm[,nn] }
```

where: mm is the number of times the student is allowed to time-out on the same QUESTION before he is branched back to review;
nn is the number of PROBLEMS that will be skipped when branching back for review; and
mm and nn must not exceed 99.

If NO is specified the tally of the number of TIMEOUTS will not be incremented and the student will be presented with the same PROBLEM until he answers it in the time allotted.

NOTE: It is the $$LATENCY DCR which determines how long a student has to answer a QUESTION before he is considered to have timed-out.

```
Example:      $$PROBLEM
              $$LATENCY  300
              $$TIMEOUTS YES,2,5
                QUESTION
```

Thus, if the student does not answer the QUESTION within 30 seconds he will have timed-out. If he times-out three times then he will be skipped back five PROBLEMS for review.

If desired, a student can be branched to the beginning of a LESSON for review. Please see the Note under $$WRONGS for further detail.

$$CWSTART

This DCR signifies that the input records following are actual Coursewriter II source cards. No editing of the following input records will be done and they will be written directly in the LOGIC DIVISION output. They will then become part of the Coursewriter output that is produced by the DATA DIVISION. An "en" opcode, or *END in cols. 1-4 must never be included in this Coursewriter input.

$$CWEND

This DCR signifies the end of the Coursewriter source input records. VAULT will then resume checking of input for valid LOGIC division input cards. If this card is encountered in the input stream without a preceding $$CWSTART DCR then a warning message will be generated and the record ignored.

```
Example:      $$PROBLEM
                QUESTION
              $$CWSTART
                  fn4 ed¬/¬/,D¬/Z¬/Z¬/ □
              $$CWEND
              $$PROBLEM
```

$$LIST

This DCR is used to specify whether or not a listing of the LOGIC input and the LOGIC output is required.

Format: $\text{\$\$LIST} \begin{Bmatrix} \text{[YES] [,YES]} \\ \text{[NO] [,NO]} \end{Bmatrix}$

where the first parameter refers to the listing of the input and the second parameter refers to the listing of the output.

If this DCR is omitted then only a listing of the input will be provided.

Exmaple: $$LIST NO,YES

In this case no listing of the input will be produced but the LOGIC division output will be listed.

$ Comments

Any card with a $ in column 1 and a blank in card column 2 will be treated as programmer comments. The card will be printed but no action will be taken by VAULT.

III. VERBS

In general, VERBS have two uses:

(1) to control the logical flow of the Coursewriter II program; and,

(2) to provide the means for accepting data from the DATA DIVISION.

The logical control verbs are PAUSE, ERASE, IF --- THEN, GO TO, and ASSIGN. They provide all the necessary information to the Coursewriter

II program and no additional information is required from the DATA DIVISION.

On the other hand DISPLAY, QUESTION, POINT, and SHOW indicate the points at which data should be provided by the DATA DIVISION. The particular VERB used will determine what type of data should be entered, and the parameters given will provide the means for controlling the input DATA. For example, the DISPLAY VERB specifies that textual data is required and the parameters prescribe the area of the CRT screen to be used.

In effect then, these VERBS in LOGIC provide a logical course structure which will be filled in by the corresponding KEYWORDS in DATA.

DISPLAY

The DISPLAY VERB requires that text data should be provided by the DATA DIVISION at this point in the program.

Format: DISPLAY [INSERT] [SR,SC,NR,NC [aaaaaa]]

SR specifies the starting row of the material to be displayed on the CRT screen.

SC specifies the starting column of the material to be displayed on the CRT screen.

NR specifies the maximum number of rows on the CRT that can be used for the display of the textual material.

NC specifies the maximum number of columns to be used for display of the textual material on the CRT screen.

aaaaaa is an alphabetic name of six characters or less that can be given and later used to refer to this set of coordinates.

INSERT specifies that textual material is to be "inserted" in text text that is already on the screen.

If no parameters are given the system will default to:

```
DISPLAY 0,0,28,40
```

The parameters can never exceed the limits of the screen, that is rows 0 to 31 and columns 0 to 39. Also rows 30 and 31 are reserved for system messages and should not be used by the programmer.

Any of the coordinates can be represented by variable names provided:

(1) the name does not exceed six characters and has no imbedded special characters.

(2) a numeric value has been previously assigned to this variable name.

(3) the numeric value does not cause the coordinates to exceed the limits of the screen.

When a DISPLAY is given, the whole screen will be erased before the text for this DISPLAY is presented. If a DISPLAY INSERT is given, only the portion of the screen specified by the coordinates will be erased before the INSERT text is presented.

An indefinite pause will also be automatically generated after each page of text associated with the DISPLAY thus allowing the student to control the pace at which data is presented. There will be no automatic pauses generated for a DISPLAY INSERT.

If the programmer wishes a different action to be taken then he must supply the appropriate PAUSE and ERASE VERBS himself.

Example:

```
$$PROBLEM
  DISPLAY 10,0,10,40
```

This will cause textual material supplied by DATA DIVISION to be displayed starting in row 10 and column 0 for 10 rows and 40 columns.

ERASE

This VERB specifies that all, or a certain portion of the CRT screen is to be erased.

Format: $$\text{ERASE} \begin{Bmatrix} \text{[SR,NR]} \\ \text{[aaaaaa]} \end{Bmatrix}$$

SR specifies the starting row of the area on the CRT screen to be erased.

NR specifies the number of rows to be erased on the CRT screen.

aaaaaa specifies the label name to be erased.

This label name must correspond to a label which was previously defined in a DISPLAY verb and the coordinates given in that DISPLAY will be used to define the area of the screen to be erased.

If no parameters are present on this record then the total screen will be erased.

Example:

```
$$PROBLEM
  DISPLAY INSERT  10,0,4,8
  PAUSE  50
  ERASE  10,4
  .
  .
  .
$$PROBLEM
```

This would cause the material displayed to be erased after five seconds.

NOTE: Under current VAULT implementation, if a label is used with ERASE when that label must have been defined in the immediately preceding VERB.

PAUSE

This VERB specifies that the execution of the program shall pause for a certain length of time.

Format: PAUSE [tttt]

tttt specifies the number of tenths of seconds that will pass before execution is resumed. The range of the parameter is 0 to 9999.

The parameter can also be a variable providing it is:

(1) no longer than six characters;

(2) has no imbedded special characters;and,

(3) has been previously assigned a value within the allowable numeric range.

If no parameter is specified for this VERB then execution will be halted indefinitely until the student depresses a key on the keyboard.

Example:

```
$$PROBLEM
  DISPLAY
  PAUSE  50
  QUESTION
$$PROBLEM
```

This will cause a pause of 5 seconds between the presentation of textual material and the question presentation.

QUESTION

The QUESTION VERB is used to supply the necessary parameters so that the teacher can present the student with a question, accept his answer and issue a reinforcement or correctional message.

Format:

```
QUESTION[INSERT] [SR1,SC1,NR1,NC1/SR2,SC2,NR2,NC2/SR3,SC3,NR3,NC3]
```

INSERT specifies that the question text is to be inserted on a screen that already has text displayed on it.

The first set of parameters (those ending in 1) refer to the area on the CRT screen where question textual material is to be displayed.

The second set of parameters (those ending in 2) refer to the area on the CRT screen where the student response is to be entered.

The third set of parameters (those ending in 3) refer to the area on the CRT screen where a reinforcement message is to be given.

SRn specifies the starting row of area n
SCn specifies the starting column of area n
NRn specifies the number of rows in area n
NCn specifies the number of columns in area n

If no parameters are given the system defaults are:

```
QUESTION  6,0,10,40/17,0,4,40/22,0,6,40
```

Previously assigned variables may be used to specify any of the coordinates provided the variable name conforms to the rules for names.

Also any of the coordinates may be omitted and the system defaults will be used instead.

Example:

```
QUESTION  2,,,5/,3 would be the same as
QUESTION  2,0,10,5/17,3,4,40/22,0,6,40
```

Example:

```
$$PROBLEM
  DISPLAY
  QUESTION  2,0,10,5/17,3,4,40/22,0,6,40
  :
  :
$$PROBLEM
```

This example will display textual material then pause until the student is ready to continue. The student will then be presented with QUESTION text in rows 2 to 11; his answer will be accepted in rows 17 to 20 and a reinforcement message will be given in rows 22 to 27.

POINT

The POINT VERB is used to supply the necessary parameters so that the teacher can present the student with a question, accept a light pen response and issue a reinforcement or correctional message.

Format: POINT [SR1,SC1,NR1,NC1/SR2,SC2,NR2,NC2]

The first set of parameters (those ending in 1) refer to the area on the CRT screen where question textual material is to be displayed.

The second set of parameters (those ending in 2) refer to the area on the CRT screen where a reinforcement message is to be given.

SRn specifies the starting column of area n
SCn specifies the starting column of area n
NRn specifies the number of rows in area n
NCn specifies the number of columns in area n

If no parameters are given the system defaults are:

POINT 0,0,6,40/22,0,6,40

Previously assigned variables may be used to specify any of the coordinates provided the variable name is alphameric and not more than six characters.

Also any of the coordinates may be omitted and the system defaults will be used instead.

The POINT VERB is used in conjunction with the DISPLAY INSERT in the following manner. The POINT will provide the parameters for the question text and a number of labelled DISPLAY INSERTS are given to define areas where different light pen responses can be received. Separate labelled text will be given in DATA for each DISPLAY INSERT thereby providing a set of multiple choice answers for the student to choose from.

Example:

```
$$PROBLEM
  ERASE
  DISPLAY INSERT 10,0,2,40 A
  DISPLAY INSERT 13,0,2,40 B
  DISPLAY INSERT 16,0,2,40 C
  DISPLAY INSERT 19,0,2,40 D
  DISPLAY INSERT 22,0,2,40 E
  POINT   0,0,8,40/25,0,2,40
```

In this case the area from row 0 to row 7 will be used for the question text. The areas labelled A,B,C,D,E, will be used to display separate multiple choice answers and a light pen response can be accepted from any one of these areas. The reinforcement message will be given in rows 25 and 26.

SHOW

This VERB gives the necessary parameters for using the Image Projector. The Image Projector can be controlled completely from the LOGIC DIVISION or else one of the parameters may be overridden in the DATA DIVISION.

Format: SHOW [[CLOSED,]FFFF[,nnnn]]

CLOSED specifies that the film is to be positioned only and that the shutter is to remain closed.

FFFF specifies the film frame number that is to be projected and must be in the range 1-1022 (this parameter may be overridden in the DATA DIVISION).

nnnn specifies the number of tenths of seconds for which the frame is to be exposed and must be in the range 0 to 9999.

If there are no parameters present, then the system will assume that the frame at which the Projector is currently positioned is to be projected for an indefinite amount of time. If the word CLOSED is not present, then it is assumed that the shutter should be opened.

The third parameter, (if any), will be ignored if CLOSED is used. If CLOSED is the only parameter then the shutter of the Image Projector will be closed and no film positioning will be done.

GO TO

This VERB is used to control the logical flow of the program. It specifies that an unconditional branch should be taken.

Format: GO TO [label]

The label must be a maximum of six characters in length beginning with an alphabetic and having no embedded special characters. This label must match precisely with one specified with a $$BLOCK, $$LESSON, or $$PROBLEM.

When the DATA DIVISION is compiled, the instruction will then be modified to branch to the label of the BLOCK, LESSON or PROBLEM in DATA which corresponds to the one referred to in LOGIC. The instruction will be further modified if the number of subdivisions (BLOCKS, LESSONS, PROBLEMS) in the DATA DIVISION is different from that in the LOGIC DIVISION.

Example:

```
           GO TO PROB2
           :
          $$PROBLEM PROB2
```

When the Coursewriter II program is executed it will branch to the PROBLEM that was associated with PROB2 in LOGIC. If no DATA is supplied for PROBLEM PROB2 then a dummy label will be created to accommodate the branch.

IF --- THEN

This VERB specifies that if the specified condition is true, during execution of the program, then the specified action will be taken.

```
Format:        IF (condition)
               THEN (action)
```

The condition can be a logical comparison of counters versus numbers or variables, or it could be a description of a possible student anser. The allowable conditions are:

```
CORRECT
WRONG
UNREC
TIMEOUT
ADDITIONAL
CORRECTS (relation) nn
WRONGS (relation) nn
UNRECS (relation) nn
TIMEOUTS (relation) nn
ADDITIONALS (relation) nn
TIME (relation) tttt    [COUNTER-CC]
COUNTER-CC (relation)   [xxxxx]
SWITCH-SS (relation) y
```

```
where:  (relation) is either >,<,=,≠,>=,<=
        nn is a number from 0-99
        tttt is the number of tenths of seconds in the range 0-9999
        CC is the counter number from 0-30
        xxxxx is a number in the range -32768 to +32767
        SS is the switch number from 0-31
        y is 0 for OFF
             1 for ON
```

The action to be taken by the THEN must be GO TO or STOP with the necessary parameters.

The THEN section of this VERB must be on a separate input record from the IF section.

Example 1:

```
IF CORRECT
THEN GO TO START
```

Example 2:

```
IF COUNTER-4 = 25
THEN STOP
```

STOP

The STOP VERB is used to halt execution of the program at any point in the program. There are no parameters associated with this VERB.

Format: STOP

Its main use will be with the IF --- THEN VERB to halt execution if a certain condition occurs.

Example:

```
IF WRONGS = 20
THEN STOP
```

ASSIGN

This VERB is used only during LOGIC DIVISION compilation and is not used to provide information directly to the DATA DIVISION. It allows an expression to be evaluated and the value assigned to a particular variable. The variable can then be specified as a parameter, and its assigned value will be extracted and used.

Format: ASSIGN [expression-1 [expression-2----]]

The expressions are mathematical expressions using either the add (+), subtract (-), multiply (*) or divide (/) operators. The right hand side of the expression can have no more than 2 operands. The expression will be evaluated and stored for future use in the LOGIC compilation.

Example: ASSIGN TIME=SECS*60

This will evaluate the value of the variable SECS multiplied by 60 and store it as the variable TIME.

If the expression has no equal sign

(i.e., ASSIGN PROB)

then that variable will be deleted. If no expressions are given, then all variables currently defined will be deleted.

The variable name must begin with an alphabetic character and have no imbedded special characters. It's length cannot exceed six characters. The evaluated expression must be in the range -32768 to +32767.

IV. Maintaining Student Performance Information

Counters

Counters 0-15 and 25-30 are used in the Coursewriter II program produced by VAULT and normally cannot be used by the programmer. However, counters 1-15 can be made inoperative by the use of:

```
$$CORRECTS  NO
$$WRONGS    NO
$$UNRECS    NO
$$TIMEOUTS  NO
```

These counters will then be available for programmer use. Counters 0, 13, and 25-30 should not be altered by the programmer.

Return Registers

Return Register 1 and 2 are the only return registers used by the Coursewriter II output from VAULT. At the start of each PROBLEM, the address (label) of that PROBLEM is stored in return register 1 and the last executed PROBLEM label in return register 2. These addresses may then be used by the programmer for any branching he may want to do in Coursewriter. At present VAULT does not provide any facilities for making effective use of this address.

Buffers

Buffer 5 contains the students name if his name was used when he was registered.

Switches

Switch 31 is turned on if the student has started the course before.

Specific Counters Used

0	Time in tenths of seconds the student took to respond to a request for keyboard or light pen entry.	
1	TIMEOUTS	Number within current PROBLEM
2		Total number within current LESSON
3		Total number within current BLOCK
4		Total number in the course
5	CORRECT	Total number within the current LESSON
6		Total number within the current BLOCK
7		Total number in the course.
8	Total number of UNRECOGNIZABLE answers within the current PROBLEM	
9	Total number of WRONG answers within the current PROBLEM	
10	UNRECOGNIZABLE and WRONG	Total number within current LESSON
11		Total number within current BLOCK
12		Total number in the course

13 the rank value of the student response within a set of ADDITIONAL answers as specified in the DATA DIVISION.

14 RESERVED FOR FUTURE USE
15 RESERVED FOR FUTURE USE

16-24 Available for programmer's use

25 Total number within the current LESSON (not including TIMEOUTS)
26 attempts Total number within the current BLOCK (not including TIMEOUTS)
27 Total number in a course (not including TIMEOUTS)

28 Total number presented in a course
29 PROBLEMS Total number presented within the current BLOCK
30 Total number presented within the current LESSON

V. VAULT LOGIC Error Messages

All error messages in VAULT are given a four character number. The first character E, or W, indicates the severity of the error and the general action taken by VAULT. The last three characters provide a reference number to look up the appropriate explanation and correction procedure.

E-Level (Error)

This type of error is sufficient to cause total suppression of all the following output. Error checking will normally continue however on all the remaining input. NO LOGIC output will be produced.

W-Level (Warning)

When this error is encountered processing will continue. Depending on the nature of the error, the input card will either be ignored completely or else VAULT will try to correct the condition by assuming certain default conditions. Therefore, the user should check carefully both the input card and the reference manual to be sure he will get the results he expects. The error message will indicate exactly what action VAULT has taken on a particular error.

W-Level Errors

W010 A blank card has been encountered in the input cards. The card will be by-passed but the programmer should check his input carefully to ensure that he did not intend to put a valid LOGIC card in this position.

W020 More than 9999 LOGIC cards have been processed. No further input cards will be processed and LOGIC will be terminated in the normal way for those cards already processed.

W022 An end-of-file condition was raised on the input file before $$ENDLOGIC card was read by VAULT. The same action will be taken as if a $$ENDLOGIC had been read at this point. The programmer should check to make sure all the input cards he intended to include were actually processed.

W033 A $$CWEND was encountered without a preceding $$CWSTART with which to associate it. The $$CWEND will be ignored and processing will continue normally.

W035 The label in a $$BLOCK, $$LESSON or $$PROBLEM exceeds six characters. The label will be truncated on the right and processing continues.

W038 $$COMPTYPE is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W039 $$KEYL is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W041 $$LOGICNAME is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W042 $$ORDER is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W043 $$OUTPUT is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W044 $$PHONETICS is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W050 $$SPELLING is not yet available in VAULT LOGIC. The card is ignored and processing continues.

W063 There are more than two parameters in a DISPLAY verb not including INSERT or GRAPHIC. (note also that the coordinate set counts as one parameter for counting purposes). All other parameters then will be ignored and processing will continue.

W066 The label on the DISPLAY card is more than six characters in length. Therefore it will be truncated on the right and only the first six characters used.

W067 GRAPHIC is not yet available as a DISPLAY parameter. This DISPLAY card is ignored and processing continues with the next input card.

W080 There are more than five parameters on an input card. Check the card to be sure that unnecessary blanks are not imbedded in the parameter set. The action that will be taken by VAULT LOGIC with regard to using or ignoring the extra parameters will depend upon the particular VERB involved.

W090 The variable name of a set of coordinates is more than six characters in length. The name is truncated on the right and the first six characters used.

W095 A given variable name is more than six characters. It will be truncated on the right to six characters.

W100 The label in the POINT card is more than six characters in length. The label is truncated on the right to six characters and processing continues.

W110 In an ASSIGN card the variable that is specified to be deleted can not be found in the variable table. That is, it has not been previously defined. The variable will be ignored and processing will continue with the next expression on the ASSIGN card.

W191 The parameter in the IF statement is an expression rather than a single word but the first word of the expression does not end in "S". "S" is added to the first word and normal processing continues.

W195 There is only one "parameter" in an IF card and it is a single word rather than an expression. However, this parameter ends in an "S". The "S" is removed from the end of the parameter and normal processing continues.

W199 The argument in an IF TIME statement is not in the range 0-9999.

W221 A label of more than six characters is specified in a GO TO statement. The label is truncated on the right and normal processing continues.

E-Level Errors

E030 The DCR in the input card is invalid. Either it has been spelled incorrectly or it has not yet been made available for use in LOGIC.

E031 No parameter was found in one of the following DCR's: LIST, LATENCY, CORRECTS, SPELLING, TIMEOUTS, UNRECS, WRONGS. A parameter is required for each of these DCR's and must be included.

E032 There were too many parameters in one of the following DCR's: LATENCY, CORRECTS, TIMEOUTS, UNRECS, WRONGS, or else one of the parameters that was there was incorrect.

E034 A $$DATA was encountered and this is not valid in the LOGIC DIVISION. This $$DATA is treated as a $$ENDLOGIC, and the LOGIC division compilation is terminated immediately.

E035 More than 200 labels have been generated for this particular BLOCK which is being processed and the system can not handle any more. LOGIC is terminated immediately.

E039 A label was encountered that has already been used in a previous DCR. This is an illegal condition. Please check the complete program for any branches to this label and correct all invalid references.

E036 A LESSON number greater than 99 has been encountered for this BLOCK. As the maximum is 99, the system cannot handle this LESSON number and LOGIC is terminated immediately.

E037 A PROBLEM number greater than 99 has been encountered for this particular LESSON. As the maximum is 99, the system cannot handle this PROBLEM number and LOGIC is terminated immediately.

E040 Either there was no parameter on a $$LIST card or else the parameter was not YES or NO.

E045 One of the following rules for the parameter in a LATENCY card has been violated: the parameter is a variable name of more than six characters, the variable has not been previously assigned, or the value of the variable is not in the range 0-9999.

E046. A LATENCY card is not preceded by a $$PROBLEM (either actual or implied as with the first PROBLEM in the course).

E048 The parameter in one of the following DCR's ends with a comma; CORRECTS, WRONGS, UNRECS, TIMEOUTS. This is not legal syntax and therefore the card cannot be processed.

E049 The parameter in CORRECTS, WRONGS, TIMEOUTS, or UNRECS is either not numeric or it is not in the range 1-99 if it is numeric. This is an illegal condition and therefore no further output will be produced.

E051 A CORRECTS, WRONGS, TIMEOUTS or UNRECS is not preceded by a PROBLEM, LESSON or BLOCK DCR. No further output is produced. Syntax checking continues.

E061 A VERB has been encountered which is not valid. Either it has not been made available for use yet or it is spelled incorrectly.

E065 More than 4 coordinates have been specified for a DISPLAY verb. LOGIC cannot handle this invalid condition so the card is not processed and no further output is produced.

E066 The first character of the label associated with the DISPLAY verb is not alphabetic. Therefore, the label cannot be processed.

E070 The label in an ERASE verb does not match the label in the immediately preceding DISPLAY card which gives the coordinates to be erased. This card can therefore not be processed as no coordinates are available for its use.

E071 Only one coordinate is specified in an ERASE card. Both coordinates must be specified if coordinates are being used rather than a label.

E080 An "en" card or *END label was found in user Coursewriter II input. As this will terminate the program when it is run on the IBM 1500 it is assumed that the programmer did not intend to include it.

E090 More than two parameters were found in a QUESTION card (not including "INSERT"). Note that the set of coordinates counts as one parameter and therefore there should be no imbedded blanks in the coordinate set.

E091 For a particular card type, the set of coordinates given do not fall within the allowable range. Check the rules for the particular VERB in question and correct the coordinate values accordingly. In most cases syntax checking of input will continue but no further output will be produced.

E095 For a particular card type the value of a parameter does not fall within the specified range. Check the rules for the particular card type which caused the error to see what the range should be.

E096 A variable given in an input card has not been previously defined and therefore a required value cannot be obtained for an input card. In most cases syntax checking of input will continue but there will be no further output produced.

E097 The value previously assigned to a variable does not fall within the limits for the parameter in question for a particular card type. Check the rules for parameters for the card which caused the error. In most cases syntax checking of input will continue but there will be no further output.

E100 An undefined variable has been used in an expression to be evaluated by an ASSIGN and therefore, the expression cannot be evaluated.

E101 The numeric value to be assigned to a variable by ASSIGN exceeds the range ± 32,767. As this is the maximum number allowed, the ASSIGN cannot be completed.

E102 The programmer has tried to ASSIGN values to more than 50 variables. As 50 is the maximum number that can be handled by VAULT LOGIC, this ASSIGN cannot be processed. The programmer should delete any unnecessary variables.

E103 An attempt was made to divide by zero when evaluating an expression in an ASSIGN card. This cannot be carried out and the programmer should check the expression and variables involved and rectify the problem. This error will be raised also if the evaluated expression is not in range -32768 to 32767.

E105 An imbedded blank was found in an expression to be evaluated by an ASSIGN card. This cannot be handled and therefore the programmer should check the syntax rules for ASSIGN expressions carefully and rectify the problem.

E190 No parameters were found in an IF card. As this makes the IF meaningless no action can be taken by VAULT LOGIC.

E192 The parameter in the IF card was an expression but the first word in the expression was not one of the following TIME, ADDITIONALS, WRONGS, UNRECS, CORRECTS, TIMEOUTS and it was also not a COUNTER, BUFFER or SWITCH. As no other parameters are valid the IF card could not be processed.

E193 The BUFFER, COUNTER or SWITCH number specified in an IF statement was not within its required limits that is 0-5 for BUFFER, 0-30 for COUNTER, and 0-31 for SWITCH. The IF card can therefore not be processed.

E195 A single word parameter was detected in an IF statement but it was not one of the following: ADDITIONAL, WRONG, TIMEOUT, UNREC, CORRECT. No other single word parameters are allowed so the IF statement can not be processed.

E196 The right hand argument of an IF SWITCH is either COUNTER, BUFFER or SWITCH which are all invalid.

E197 The right hand argument of an IF COUNTER is either BUFFER or SWITCH which are invalid.

E198 COUNTER number of right hand argument of an expression exceeds the limits 0-30.

E199 The right hand argument of BUFFER is not BUFFER.

E200 A THEN statement has been encountered with no preceding IF statement. As THEN is meaningless in this context it cannot be processed.

E201 An IF statement has already been processed but there is no THEN statement following it. This is an illegal situation and makes the IF statement unusable.

E220 In a GO TO statement either "TO" is missing or else the label to which the program should branch is missing.

E221 The first character of a label in a GO TO statement is not alphabetic. This not a valid label and therefore the GO TO cannot be processed.

E222 200 branch (GO TO etc.) statements have already been encountered in this run of VAULT LOGIC. As this maximum of 200 cannot be exceeded this GO TO statement cannot be processed.

E250 Either the parameter in a PAUSE statement is a label which does not have an alphabetic character for the first character, or, the actual number specified is greater than 9999.

E251 The variable specified in the PAUSE card cannot be found in the variable table or if it was found its value is not in the range 0-9999.

E900 A $$BLOCK card was encountered after another major DCR, e.g., $$LESSON or $$PROBLEM. The sequence of the input cards should be checked carefully.

APPENDIX E

VAULT DATA DIVISION SPECIFICATIONS

TABLE OF CONTENTS FOR APPENDIX E

I. Division Control Records (DCR)

These records have three functions:

(1) to logically subdivide the program into BLOCKS, LESSONS, PROBLEMS;

(2) to provide the IBM 1500 Instructional System with the necessary information to execute the program, and;

(3) to signify the beginning or end of the program.

All DCR's begin with two dollar signs ($$).

Following the name, at least one blank space is required after which the desired parameters may be punched. The parameters may begin anywhere after the name and a blank, but must end on or before card column 71.

Any information punched in card columns 73-80 will be ignored to allow this area to be used for card sequencing.

Certain DCR's may be omitted. If this is done, a standard set of options which is associated with each DCR will be automatically provided by VAULT.

Three of the DCR's ($$BLOCK, $$LESSON, $$PROBLEM) referred to as "major" DCR's, are matched against the incoming LOGIC to modify presentation. It is not necessary that these DCR's conform either in number, or parameters with the LOGIC.

Any labels that are used (see discussion of labels under specific DCR's) must begin with an alphabetic character, and be not more than six characters in length. The characters other than the first can be alphabetic or numeric.

$$DATA and $$ENDATA signify the beginning and the end of the program respectively and should not be omitted.

All the DCR's listed below are optional and are not compared in any way with the LOGIC DCR's.

$$AUTHOR
$$COURSENAME
$$LIST
$$STARTBLOCK
$$STARTLBL
$$LOGPK
$$ROWSIZE
$$HYPHEN
$$CWDECK

Major DCR's

$$DATA

This signifies the beginning of the input to the VAULT DATA DIVISION. It should normally be the first input record. All card columns following $$DATA are ignored by VAULT and may be used simply for comments by the author.

Format: $$DATA [comments]

Example: $$DATA THIS IS THE BEGINNING OF TRIGONOMETRY

NOTE: If a parameter is enclosed in square brackets, i.e., [comments], then it may be optionally included or omitted by the programmer.

$$ENDATA

This DCR signifies the end of the input cards. It should be the last card since any cards following will be completely ignored. Any information punched following $$ENDATA will be ignored by VAULT and may be used for comments.

Format: $$ENDATA [comments]

Example: $$ENDATA THIS IS THE END OF TRIGONOMETRY

NOTE: $$ENDATA has only one "D."

$$BLOCK

This DCR signifies that all following material is to be treated as a logical grouping on the IBM 1500, much like a chapter in a textbook. This record may be omitted.

This BLOCK (chapter) may be given a one-word label (name). Anything further on the card will be disregarded.

Only one $$BLOCK record may be compiled at one time and it must precede the other major DCR's and the keywords. More than one $$BLOCK record will cause VAULT to stop further processing and to output any Coursewriter II code that has already been generated.

A BLOCK may contain as many as 99 LESSONS or as few as one.

Format: $$BLOCK [label]

Example: $$BLOCK MATH10

$$LESSON

This DCR signifies that the following input up to the next $$LESSON is to be treated as a logical subdivision of the BLOCK. The first $$LESSON record may be omitted.

As many as 99 LESSONS are allowed, each one of which may contain from 1 to 99 PROBLEMS. If more than 99 $$LESSON cards are encountered, VAULT will stop processing and any Coursewriter II code already generated will be output.

When this record is encountered, the LOGIC is checked for a corresponding LESSON DCR. If it is at a LESSON DCR already, then VAULT will proceed normally. Otherwise, intervening LOGIC will be skipped until a LESSON DCR is encountered. If there are no further LESSONS in LOGIC, then VAULT will go back to the beginning of the LOGIC BLOCK and continue processing from the first LESSON in the BLOCK (chapter).

The LESSON may also be labelled (i.e., named) if desired by supplying a label on the record.

Format: $$LESSON [label]

Example: $$LESSON PART1

$$PROBLEM

This DCR signifies that the following input to the next $$PROBLEM is to be treated as a logical subdivision of the LESSON. The first PROBLEM within a LESSON does not require a $$PROBLEM record.

As many as 99 PROBLEMS may be entered in any LESSON. If more than 99 are encountered, then VAULT will stop processing records within that LESSON. If there are no further LESSONS in this DATA, then all the Coursewriter II code already generated will be output.

If more PROBLEMS are supplied in the DATA DIVISION than exist in the LOGIC, VAULT will return to the beginning of the current LOGIC LESSON and continue processing DATA from the first PROBLEM in that LESSON. In this way, the DATA DIVISION can have more , as many, or less PROBLEMS than in the corresponding LOGIC.

The PROBLEM may also be labelled by punching a label, or name, on the card.

Format: $$PROBLEM [label]

Example: $$PROBLEM PRO15

Leading DCR's

$$CWDECK

It is probable that a teacher would want to only have a printed listing of his program output at first until he is sure that all coding errors have been removed from his input. Until such time he would not be ready to try his program out on the IBM 1500.

This DCR, $$CWDECK, is designed to permit the production of Coursewriter II cards. This record may be omitted, in which case a deck will not be produced.

If the author wishes to have a deck produced he should punch YES as the parameter. If he does not yet require a Coursewriter II deck he should either omit the card or enter the parameter NO.

Format: $$CWDECK $\left\{\begin{matrix}\text{[YES]}\\ \text{[NO]}\end{matrix}\right\}$

Example: $$CWDECK NO

This would cause suppression of punching of Coursewriter II cards.

$$AUTHOR

This record specifies the author "name" for the IBM 1500 Instructional System. The name must begin with the letter A through I and be 4 or less characters in length.

This record may be omitted, in which case VAULT will use A001 as the author "name."

Format: $$AUTHOR XXXX

Example: $$AUTHOR D176

$$COURSENAME

This record specifies the name of the course that is being programmed. The name should be five characters or less in length. It must begin with an alphabetic character, and the remaining characters must be either alphabetic or numeric.

This record may be omitted, in which case VAULT will use one supplied in the LOGIC. However, as this is the name that will be used when calling the program for execution on the 1500 system, this DCR should be included to ensure a unique course name.

Format: $$COURSENAME XXXXX

Example: $$COURSENAME MATH

$$LIST

This DCR specifies whether or not a printed listing of the Coursewriter II code is desired. If a listing is desired, punch YES on the card anywhere following $$LIST. If a listing is not desired, punch NO. This record may be omitted, in which case a listing will be produced.

Format: $$LIST $\left\{\begin{matrix}\text{[YES]}\\ \text{[NO]}\end{matrix}\right\}$

Example: $$LIST NO

(This would cause VAULT to NOT list the Coursewriter II code.)

NOTE: $$LIST does not effect error messages. They will always be printed.

$$STARTBLOCK

Since the BLOCK is like a chapter in a textbook, this DCR supplies a number to the chapter (BLOCK).

If this record is omitted, the number assumed is 1 (i.e., chapter 1).

```
Format:         $$STARTBLOCK  XX
```

where: XX is a number between 1 and 99 inclusive.

```
Example 1:      $$STARTBLOCK   3

Example 2:      $$STARTBLOCK  15
```

NOTE: If $$STARTLBL (see page 214) is also used and the label specified is numeric, (e.g., $$STARTLBL 150304) then the first two digits must equal the STARTBLOCK number (chapter number).

The following example is correct.

```
$$STARTBLOCK 10
$$STARTLBL   100514
```

The following example is incorrect.

```
$$STARTBLOCK  10
$$STARTLBL    150309
```

$$STARTLBL (Start Label)

It will often occur that a course or BLOCK is only partially completed one day and processed by VAULT. Later on, an author could produce additional VAULT code which he wants processed. In this case, it would be necessary to provide VAULT with the last label used. The $$STARTLBL does this.

The label given must already exist in the COURSEWRITER II program to which the teacher wishes to add. If the label is numeric then any further labels generated by VAULT will sequentially follow the label given in the $$STARTLBL card. If the label is alphabetic then VAULT would produce labels stating at a $$STARTBLOCK parameter (01), LESSON (01), and PROBLEM (01). Note that the DATA input will be matched with the beginning of the LOGIC program.

$$LOGPK

This DCR specifies the number of the IBM 1500 disk pack on which an author desires this course to be stored. The number given must be between 0 and 32767 inclusive. It is suggested that an author check with the IBM 1500 System Manager before using this DCR. If this input record is omitted then the logical pack number assumed will be 00000.

```
Format:        $$LOGPK  XXXXX
Example 1:     $$LOGPK  1043
Exampel 2:     $$LOGPK     0
```

Optional DCR's

$$ROWSIZE

This DCR allows an author to specify what kind of spacing is desired on the 1500 Display Screen (I.E., single spacing, double spacing, etc.) The number given normally has limits of 2 through 28. If this DCR is omitted, the system will use $$ROWSIZE 2, i.e., single spacing. A particular ROWSIZE will remain in effect until another ROWSIZE DCR is encountered in DATA.

```
Format:        $$ROWSIZE  XX
```

Example 1: $$ROWSIZE 3

Example 2: $$ROWSIZE 4

NOTE: The IBM 1500 requires two rows to display one line of textual material. Therefore, the following record is invalid:

$$ROWSIZE 1

$$HYPHEN

This DCR permits the author to specify the minimum size of word that may be hyphenated (broken) when a word is too long to fit into the number of columns that remain on the current row of the CRT. If this record is omitted the system default will break a word only if it contains at least 6 characters. A restriction is that the parameter may not be as large as or larger than the screen width defined by the logic parameters.

Format: $$HYPHEN XX

Example 1: $$HYPHEN 8

Exampel 2: $$HYPHEN 20

II. Keywords

Keywords have four uses:

(1) to provide curriculum material that is to be presented to the student.

(2) to provide textual material that is to be presented in the form of a question to the student.

(3) to specify expected answers of the student (i.e., correct answers, wrong answers, etc.)

(4) to provide corrective or reinforcement material that is to be presented to the student depending on his type of answer.

Basically, any portion of a card may be used from card column 1 through 71. The keyword, however, must be the first word punched on the card and it must be completed by card column 71. All keywords should be followed by a colon (:) and one blank space.

Following is a list of VAULT keywords.

TEXT:
QUESTION:
CORRECT:
WRONG:
ADDITIONAL:
TIMEOUT:
UNRECOGNIZABLE: or UNREC:
SHOW:

Since the LOGIC that has been provided has already defined the particular devices to be used, the order in which they are used, etc., it is required that the DATA input be somewhat conformable to the specified LOGIC. For example, if the LOGIC calls for material to be displayed on the 1500 Display Screen, it is necessary that correspondingly the DATA input provide the material to be displayed. If the LOGIC expects some material to be presented in the form of a question, then the DATA must, at that time, provide the question material.

Necessarily then, the teacher must be aware of the requirements of the LOGIC, and adapt his subject matter input correspondingly.

There are eight keywords available to the teacher. Two of these, TEXT: and QUESTION:, must be conformable in position to LOGIC. The keyword SHOW: is optionally conformable. The others are not matched in any way to the LOGIC.

Following is a list of the keywords by conformity:

conformable	optionally conformable	non-conformable
TEXT:	SHOW:	TIMEOUT:
QUESTION:		CORRECT:
		WRONG:
		ADDITIONAL:
		UNRECOGNIZABLE:
		UNREC:

Conformable Keywords

TEXT:

The purpose of this keyword is to provide textual material which will be presented to the student on his Display Screen.

Since the precise placing of text on the screen is defined in LOGIC, the relative position of this keyword within a PROBLEM must be the same as that within the LOGIC PROBLEM. A short look at the LOGIC will tell one where to place this record.

The textual material is expected to begin following the keyword, a colon and a space. The material must end with an end-of-text character (i.e., 0-2-8 punch). This is normally signified in writing by a small box (□).

```
Format:     TEXT: [Display material]□

Example:    $$PROBLEM
            TEXT: <I>N 1534, <J>ACQUES <C>ARTIER SAILED FROM
            <S>T. <M>ALO IN <F>RANCE.□
```

Note the characters surrounding the I in "IN," the J in "JACQUES," the C in "CARTIER," etc. These characters, and other are necessary for the computer. As one can check, the keypunch only punches capital letters, unlike a normal typewriter or the IBM 1500 Display Terminal.

It is necessary therefore to tell the computer when a capital is desired, and when a lower case is desired. Furthermore, certain characters on the keypunch keyboard are differently placed from the Display Keyboard. The difference also requires special keypunching techniques. (For example, note the "*", the "%", etc.) Five special keypunching requirements should be noted:

(1) new paragraph character--"*"

This character (*) indicates the following material is to begin a new paragraph.

```
Example:        TEXT: <F>OLLOWING ARE THE TOPICS WE WILL DISCUSS TODAY:
                **SINE*COSINE*TANGENT.□
```

This will cause the following material to be presented to the student:

```
                Following are the topics we will discuss today:

                sine
                cosine
                tangent
```

(2) new paragraph beginning on the next card--"¬*"

These two characters together (¬*) signify that the material beginning on the next card is to begin a new paragraph, i.e., the remainder of the present card will be disregarded.

```
Example:        TEXT: <TYPES OF ANGLES>¬*THIS WILL NOT APPEAR
                <S>UPPLEMENTARY*<C>OMPLEMENTARY*<A>CUTE*<O>BTUSE*<R>IG
                HT□

CRT display:    TYPES OF ANGLES
                Supplementary
                Complementary
                Accute
                Obtuse
                Right
```

(3) new screen characters--"¬$"

These two characters (¬$) specify that an indefinite pause followed by an erasure of the screen is required. VAULT reads a new card and places this new material at the top of the screen area as designated by the parameters of the corresponding DISPLAY verb in logic.

Example:

```
TEXT: <F>OLLOWING THIS MATERIAL SHOULD BE A MESSAGE
TELLING THE STUDENT TO 'PRESS THE SPACE BAR TO CONT
INUE'.¬$
<T>HIS MATERIAL SHOULD NOW APPEAR AT THE TOP OF THE
SCREEN.□
```

(4) continuation of text characters--"¬#"

These two characters are primarily intended for editing punched VAULT cards. When these characters (¬#) are encountered VAULT reads a new card and continues editing the text in a manner similar to the situation when card column 71 has been processed.

Example:

```
TEXT: <T>HE R¬#
E¬#
D HORSE.□
```

The following material would appear on the CRT:

The red horse.

(5) end-of text character--"□"

This is a 0-2-8 punch (numeric "T" on the keypunch). It signifies that the end of the textual material has been reached. The remainder of this card will be disregarded.

Labelling of Text

There are times when labelling of textual material becomes desirable, particularly when working with the "light pen" rather than the student's keyboard. This is simply done by following the keyword and its colon by a space, a label, a colon, and a space (note: Do not omit the colons).

Example:

```
TEXT: HIST1: <I>N 1493 <C>ARTIER SAILED AGAIN FOR THE NEW
WORLD.□
```

This material and its section on the Display Screen may be referred to by use of label HIST1. This means that the student may point to any portion of the CRT defined by the parameters of the corresponding DISPLAY verb in logic. Please refer to sections dealing with light pen responses to determine in what way labelling becomes useful.

QUESTION:

The purpose of this keyword is two-fold:

(1) to provide the textual material that is to be presented to the student as a question, and;

(2) to request a keyboard response or a light pen response when it is expected from the student.

The corresponding LOGIC, to this point, would specify three things:

(1) the precise placing on the screen of the QUESTION, the student response (when a keyboard response is required), and any reinforcement or corrective messages;

(2) the relative position of this keyword within the PROBLEM; and,

(3) the sequence of information presentation to the student depending on his type of answer.

Since these things have been previously specified, the relative position of this keyword within the PROBLEM must be the same as that within LOGIC. A look at the LOGIC will tell one where to place this record.

Format: QUESTION: [material to be displayed]□

The general format of this record is similar to that of the TEXT: record. The question material is to begin following the keyword, a colon, and a blank. The material must end with an end of text character (0-2-8 punch). This is normally signified in writing a small box (□).

Example:

```
QUESTION: <F>ROM WHAT CITY IN <F>RANCE DID <C>ARTIER
SAIL IN 1534?□
```

The question material following a QUESTION: keyword is handled in the same manner as the material following a TEXT: keyword. Therefore, the various special characters required to signal to the computer such things as capital letters, etc., must be included in the information. (Please refer to the TEXT keyword for further information about these characters.)

NOTE: It is very important that the end-of-text characters (0-2-8 punch) not be omitted.

Light Pen Response Request

A student can respond to a question on a 1500 in one of two modes, either by typing in his answer on the display keyboard or by pointing with the light pen to the answer chosen. The particular device or method of response that is to be used is specified by the verbs DISPLAY and POINT in the LOGIC. However, in the DATA DIVISION, only one keyword (QUESTION:) can correspond to either of these two VERBS. The teacher must be aware of which type of device is called for by the LOGIC. For further detail, as to why one must be aware of what the LOGIC has called for, please refer to the section dealing with light pen responses under answer analysis.

Optionally Conformable KEYWORD

SHOW:

This keyword supplies the frame number of the film that is to be shown to the student on the IBM 1512 Film Projector. The number must range from 1 through 1022.

This keyword is optionally conformable in that the record may be omitted even though its counterpart in the LOGIC does exist. If it is omitted, then the frame number will be supplied by the LOGIC. If the record is present, however, then it must be conformable as to the relative position within the PROBLEM.

Format: SHOW: XXXX

Example: SHOW: 597

Non-conformable KEYWORDS

The keywords in this section generally provide two things:

(1) various answers anticipated by the teacher, and;

(2) reinforcement or corrective message material.

Any or all of these keywords may be omitted or used more than once. If the keywords TIMEOUT: or UNRECOGNIZABLE: are omitted, then VAULT will automatically provide the necessary Coursewriter II code to handle these situations. If the other keywords (i.e., CORRECT:, WRONG:, or ADDITIONAL:) are omitted then no Coursewriter II code will be generated for those conditions. Because of this, the teacher should be aware of what answer analysis techniques have been planned for in the LOGIC, even though these keywords do not have to conform to the LOGIC.

NOTE: If all of these keywords are omitted then any answer the student gives providing he does not timeout will be treated as UNRECOGNIZABLE.

Following is a list of the keywords available for answer analysis:

TIMEOUT:
CORRECT:
WRONG:
ADDITIONAL:
UNRECOGNIZABLE: or UNREC:

TIMEOUT:

This keyword is used to specify the message that will be given to the student if he does not answer in the time alloted to him by the LOGIC.

If the teacher wishes to provide his own corrective message for the TIMEOUT condition then this keyword should be included. If this keyword is omitted then the computer will automatically randomly select a message from a predetermined list to be displayed to the student informing him that he has taken too long to respond. Occasionally the teacher may specify that no message is to be displayed; this would be achieved by punching one blank on the TIMEOUT: record.

This keyword should normally follow a QUESTION keyword record and should normally precede the other answer analysis keywords.

Example: [a teacher supplied message]

```
TIMEOUT: <Y>OUR HAVE TAKEN TOO LONG TO RESPOND.¬*
<P>LEASE RESPOND MORE QUICKLY¬□
```

NOTE: The rules for textual presentation are the same as they are in the text keyword (i.e., the special characters for upshifting, downshifting, etc., must be included and the text must end with an end-of-text character.)

Example: [a request that no message be supplied]

```
TIMEOUT: □
```

CORRECT:

This keyword specifies all the answers that the teacher wishes to be considered as correct answers, and the message that the teacher wishes to be presented to the student if his answer matches any of those supplied and if he responds within the time alloted.

Each answer that the teacher wishes to be considered correct must be separated by a semi-colon and the last answer followed by an enter symbol. After the list of answers is completed, a reinforcement message may be specified by the teacher. If this message is omitted (two enter symbols in succession are encountered) then the computer will automatically randomly select a reinforcement message from a predetermined list to be displayed to the student. The teacher could follow the first enter symbol by a single blank and then an enter symbol; this would specify that no system message is to be generated.

This keyword should normally follow a QUESTION: and a TIMEOUT: keyword and should precede the UNRECOGNIZABLE: keyword.

The teacher has the option of specifying that any answer on any response that the student should make should be considered CORRECT. To do this the teacher need only omit the list of CORRECT answers.

This keyword requires that two end-of-text characters follow the keyword, at some point. That is to say, an end-of-text character must follow the last of the list of CORRECT answers, and one must follow the end of the reinforcement that is supplied. If any answer is to be considered correct, an end-of-text character should be in the first column following the keyword, the colon, and the blank. If the teacher wishes the computer to choose the reinforcement message, then two end-of-text characters should follow the last supplied correct answer. Note that again all the special characters that are required for the computer to handle textual information must be supplied in the reinforcement message.

Example: [a teacher supplied message]

```
QUESTION: <F>ROM WHAT CITY IN <F>RANCE DID <C>ARTIER
SAIL IN 1534?□
CORRECT: <S>T. <M>ALO□<T>HAT IS CORRECT. <L>ET'S CONTI
NUE.□
```

Example: [a system supplied reinforcement message]

```
CORRECT: <S>T. <M>ALO□□
```

Example: [suppression of any reinforcement message]

```
CORRECT: <S>T. <M>ALO□ □
```

<u>WRONG:</u>

This keyword specifies all the anticipated answers which the teacher wishes to be considered WRONG. It also specifies the corrective message that a teacher wishes to be displayed to the student should his answer match one of those in the list.

The general format is precisely the same as that for the CORRECT: keyword. (i.e., a list of specified answers separated by semi-colons, an end-of-text character, and a corrective message followed by an end-of-text character.) The teacher, similarly to the CORRECT: keyword, has the option of omitting this record, of specifying that any answer the student should make is to be considered WRONG, or to have the computer randomly select a message applicable to a WRONG answer. For details of the format please see the CORRECT keyword.

It should be noted that more than one WRONG card can be given for a question. This enables the teacher to give different correctional or reinforcement messages to the student depending upon which answer he gives.

Example:

```
QUESTION: <F>ROM WHAT CITY IN <F>RANCE DID <C>ARTIER
SAIL IN 1534?□
CORRECT: <S>T. <M>ALO□□
WRONG: <P>ARIS;<L>ONDON;<C>HERBOURG□<T>HAT ANSWER IS
INCORRECT. <L>ET'S GO BACK AND REVIEW.□
```

NOTE: It is very important that both end-of-text characters follow the keyword at some point.

UNRECOGNIZABLE: or UNREC:

This keyword is used to specify the message displayed to a student when his answer does not correspond to any of the ones anticipated by the teacher. This keyword may be omitted.

The message supplied must follow the normal rules for textual specification (i.e., include all necessary special characters and conclude with an end-of-text character).

Example:

```
QUESTION: <W>HO SAILED FROM <S>T. <M>ALO IN 1534?□
CORRECT: <J>ACQUES <C>ARTIER; <C>ARTIER; <JACQUES
CARTIER□□
ADDITIONAL:  CARTIER; JACQUES CARTIER□<Y>OU MUST C
APITALIZE PROPER NOUNS.□
UNREC: <I> DID NOT CATCH YOUR ANSWER. <P>LEASE TRY
AGAIN.□
```

Notice in this example that the judicious use of special characters in the answer lists have allowed the teacher to verify that the student did not capitalize proper names.

NOTE: If this keyword is used more than once each succeeding unrecognizable message specified by the teacher will appear each time the student answers unrecognizably to the same QUESTION. Thus, if the student has entered his third UNREC answer to the current problem then VAULT checks if at least three UNREC messages exist for that problem. If less than three UNREC messages exist then last UNREC message will be displayed; if three or more UNREC messages exist then the third UNREC message will be displayed.

Example:

```
QUESTION: <W>HO SAILED FROM <S>T. <M>ALO IN 1534?□
CORRECT: <J>ACQUES <C>ARTIER; <C>ARTIER; <JACQUES
CARTIER□□
ADDITIONAL:  CARTIER; JACQUES CARTIER□<Y>OU MUST C
APITALIZE PROPER NOUNS.□
UNREC: <I> DID NOT CATCH YOUR ANSWER. <P>LEASE TRY
AGAIN.□
UNREC: <A>GAIN <I> MISSED YOUR ANSWER. <P>LEASE CH
ECK YOUR SPELLING.□
```

In this example, the first time the student answers unrecognizably he will see the message--"I did not catch your answer. Please try again." The second and succeeding times he will see the message--"Again I missed your answer. Please check your spelling."

Light Pen Responses

The teacher should be aware of whether or not the LOGIC requested a light pen response (POINT) as opposed to a keyboard response (QUESTION). This is necessary to produce valid answer analysis.

All the answer analysis keywords are available for light pen response analysis and the general formats remain the same. However, the answers in the answer lists (i.e., those separated by colons) must be labels from immediately preceeding TEXT keywords. If labels were not used in the DATA division then the teacher must refer to the ones supplied by LOGIC.

Example:

```
TEXT: STMALO: <H>E SAILED FROM <S>T. <M>ALO IN <F>RANC
E.□
TEXT: LONDON: <H>E TOOK A BOAT TRAIN FROM <L>ONDON.□
TEXT: PARIS: <H>E LEFT <P>ARIS FOR <C>HERBOURG AND SAI
LED FROM THERE.□
TEXT: FRANCE: <H>E SAILED FROM <F>RANCE.□
QUESTION: <H>OW DID <C>ARTIER BEGIN HIS VOYAGE IN 1534
?□
CORRECT: STMALO□<V>ERY GOOD□
WRONG: LONDON;PARIS□□
ADDITIONAL: FRANCE□<C>LOSE, BUT NOT QUITE.□
```

Notice the following with regard to the above example:

(1) The answers listed after the CORRECT, WRONG, and ADDITIONAL keywords are labels associated with TEXT keywords.

(2) If the student pointed with his light pen to any portion of the area labelled STMALO he would have displayed to him the message "Very good."

(3) If he pointed to either of the areas labelled LONDON or PARIS the computer would have automatically chosen a message to be displayed to the student.

(4) If he pointed to the area labelled FRANCE he would have seen the message "Close, but not quite."

(5) The use of the end-of-text characters (□) is very important.

NOTE: The keywords TIMEOUT and UNRECOGNIZABLE are also available for light pen answer analysis.

III. VAULT DATA Error Messages

All error messages in VAULT are given a four character number. The first character, X,E, or W, indicates the severity of the error and the general action that is taken by Vault. The last three characters provide the teacher with a reference number used to look up the appropriate explanation and correction procedure.

X-Level (Terminal)

This type of error message is generated when an error is so severe as to cause immediate termination of VAULT processing. Normally, this error would be generated if VAULT itself made an error. It is unlikely a teacher would be able to recover from this error and he should convey his problem to the program authors.

E-Level (Error)

This type of error is sufficient to cause total suppression of all the following output. Error checking will normally continue however on the remaining input. No COURSEWRITER II deck will be produced if either an E-level or X-level error is encountered.

W-Level (Warning)

When this error is encountered processing will continue. Depending on the nature of the error, the input card will either be ignored completely or else VAULT will assume certain default conditions. Therefore the use should check carefully both input card and the reference manual to be sure that he will get the results he expects. The error message will indicate exactly what action VAULT has taken on a particular error.

NOTE: There are also some messages that are not associated with an error number. Instead, VAULT will produce a textual message directly on the printed listing indicating what kind of error has occurred. These error messages will generally be E-level errors but they will not be referenced in the following error documentation.

W-Level Errors

W010 A blank card was encountered in the input. The card was bypassed and no action was taken by VAULT. The teacher should remove this card and check to be sure he did not intend an actual DATA card to be in this position.

W020 More than 9999 cards have been read in the DATA input. This is the maximum number of cards that can be handled by VAULT and therefore processing will stop immediately.

W022 An end-of-file condition on the input DATA was sensed by VAULT before a $$ENDATA. This is, there were no more input cards available to VAULT. VAULT proceeds as though an $$ENDATA had been encountered and performs the necessary functions to complete processing. The teacher should check, however, to be sure all the DATA that he intended to use was included in his input cards.

W500 A DCR has been encountered by VAULT which has no parameter. Although some DCR's do not require a parameter this one must have one present. VAULT therefore, assumes a likely parameter and continues. However, the teacher might not get the results intended, therefore, this card should be checked and the desired parameter included.

W512 No colon was present directly after a KEYWORD. VAULT will check successive columns until a non-blank entry is found. If that entry is not a colon, then VAULT assumes that data from that column on is the text or parameter information it is looking for. If that entry is a colon, then VAULT checks the next column for a blank. If it is not a blank VAULT starts processing data from the first non-blank position after the colon. If it is a blank then VAULT will start processing in the next column after the first blank. The teacher should check the rules in the reference manual for this keyword carefully and be sure he adheres to them to avoide possible errors.

W541 An author name can only be made up of four characters. Therefore VAULT truncated the name on the right and only used the first four characters of the name supplied on the input card.

W555 An answer analysis card, WRONG, has been encountered by VAULT but no preceding QUESTION card was processed with which to associate the WRONG. VAULT assumes that the teacher has a definite reason for placing the card there and will produce the corresponding Coursewriter II code. However, the teacher should check carefully the sequence of his input and be sure the cards are properly placed as he intended.

W561 Only five characters for a COURSENAME are allowed by VAULT. As more were supplied by the teacher, VAULT truncated the name on the right and only the first five characters were used. The teacher should check the input rules in the reference manual carefully.

W570 An illegal parameter was entered on the $$LIST card. Only YES or NO are allowed as parameters, therefore VAULT disregarded the parameter in the input card and assumed the parameter YES.

W601 The label parameter in the $$STARTLBL card was more than six characters in length. The label was therefore truncated on the right and only the first six characters used by VAULT.

W620 An illegal parameter was entered in the $$CWDECK card. Only YES or NO are allowed as parameters, therefore, the compiler disregarded the parameter in the input card and assumed $$CWDECK NO.

W655 An answer analysis card, UNRECOGNIZABLE or UNREC, has been encountered by VAULT but no preceding QUESTION card was processed with which to associate the UNRECOGNIZABLE. The same action will be taken as for W555.

W654 An UNRECOGNIZABLE answer analysis card has already been processed for this QUESTION. VAULT will put out the Coursewriter II code for this UNRECOGNIZABLE card also, and it will be executed when the student enters his second UNRECOGNIZABLE response for this particular QUESTION. The teacher should check carefully to be sure he intended to use the message automatically produced by VAULT before he uses his own message. A more detailed explanation of how more than one UNRECOGNIZABLE answer analysis card may be used for the same QUESTION can be found in the reference manual.

W665 An answer analysis card, ADDITIONAL has been encountered by VAULT but no preceding QUESTION card was processed with which to associate the ADDITIONAL. The same action will be taken as for W555.

W675 An answer analysis card, CORRECT, has been encountered by VAULT but no preceding QUESTION card was processed with which to associate the CORRECT. The same action will be taken as for W555.

W690 There are too many PROBLEMS in the DATA division for them to be accomodated by the corresponding LOGIC. The teacher need not necessarily have used more than 99 PROBLEMS but because of repetition or branching in the LOGIC more than 99 PROBLEMS might be assumed to be present. The teacher should therefore, check the LOGIC division carefully to be sure he understands what it is doing and reduce the number of problems in DATA accordingly.

W705 An answer analysis card, TIMEOUT, has been encountered by VAULT but no preceding QUESTION card was processed with which to associate the TIMEOUT. The same action will be taken as for W555.

W704 A TIMEOUT answer analysis card has already been processed for this QUESTION. The VAULT will put out the Coursewirter II code for this TIMEOUT card also but it will never be executed when the program is run on the IBM 1500. The teacher should check carefully to see which TIMEOUT he intended to use and remove the other TIMEOUT card from his input DATA and also remove the corresponding Coursewriter II data from the output deck, if one was produced. This will eliminate the need for running VAULT again to get the desired results in the Coursewriter II program.

W790 Only 32 different display areas are allowed on the screen for detection of light pen response for any particular problem. Therefore, any further labelled areas supplied to VAULT will be disregarded and it will not be possible to detect a light pen response on such an area. The teacher should reorganize his data so that a maximum of 32 possibilities are available for light pen response for any problem.

W791 A label on a TEXT card can only consist of six characters. As more characters were present on the input card VAULT truncated the label on the right and only used the first six characters.

W801 VAULT can not automatically put out a reinforcement message to the student as the display area assigned for such a message is too small to accomodate any of those messages automatically produced by VAULT. The teacher can compensate by putting out a message of his own as long as it is small enough to fit in the prescribed area. The LOGIC division should be checked to see exactly how large a message area has been allowed for this particular response.

W802 The message supplied by the teacher is too long for the display area specified by the LOGIC parameters. Check the parameters in the corresponding LOGIC VERB and shorten the message accordingly.

W901 The above COURSEWRITER branch statement has been altered. This has resulted because less PROBLEMS have been entered in this DATA LESSON than anticipated in the corresponding LOGIC LESSON.

E-Level Errors

E510 VAULT was expecting a keyword as the first word in the input card and there was none present. The teacher should check the spelling of the keyword in his card or check the sequence of his input cards so that a proper keyword will be provided to VAULT at this point in the program.

E520 A TEXT card is required by VAULT at this point in the program to go with the corresponding DISPLAY verb in LOGIC. A TEXT keyword was not found as the first word in the input card. The teacher should check the sequence of the input cards and ensure a TEXT card is put in at this point.

E530 Too many special characters were present in a section of TEXT cards. Only two Coursewriter II cards can be used to produce one line on the display screen. However, there are so many special characters in the input that more than two Coursewriter II cards would have to be produced for one display line and this situation can not be handled. The teacher will have to reorganize his TEXT so that fewer special characters are required.

E540 The parameter in the $$AUTHOR card, that is, the author number, must begin with the letter A through I. Please check the reference manual for rules regarding this DCR.

E560 The first character of the COURSENAME must be alphabetic. Also no special characters are allowed anywhere in the COURSENAME. As these rules were not adhered to by the teacher, the reference manual should be checked to ensure the $$COURSENAME parameter entered correctly.

E580 The $$STARTBLOCK parameter either was not numeric, or if it was numeric, it was not in the rnage 0-99. As not more than 99 BLOCKS are ever allowed by Coursewriter II the STARTBLOCK number must be in this range.

E581 Either, more than one $$STARTBLOCK DCR was found in the input DATA or else the $$STARTBLOCK was not encountered before the first KEYWORD. Either of these conditions will cause an error in VAULT. Check the reference manual for rules regarding $$STARTBLOCK.

E582 A $$STARTLBL with a numeric label has already been processed by VAULT. The first two digits, which indicate the starting BLOCK number, do not match the number in the $$STARTBLOCK card just read. The parameters in these two DCR's should be reconciled.

E590 The parameter in the $$LOGPK card is not numeric. Check the reference manual for restrictions on $$LOGPK.

E591 The pack number on the $$LOGPK is not in the range 0-32767 as is required. Check the reference manual for rules regarding the use of this DCR.

E600 Only one $$STARTLBL DCR is allowed in a run of VAULT and it must be placed before the first KEYWORD in the program. As these rules were not followed, an error occurred in VAULT.

E603 A $$STARTBLOCK DCR has already been processed by VAULT. The parameter in the $$STARTLBL card just read does not match the BLOKC number. The reference manual should be checked for the relationship between these two DCR's and the parameters reconciled accordingly.

E610 The parameter in the SHOW card was not numeric as required. Check the reference manual for SHOW restrictions.

E611 The parameter in the SHOW card was not in the range 1 to 1022 as required for the film projector.

E630 VAULT expects a QUESTION keyword at this point in the course and one was not found in the DATA input. The teacher should be sure his cards are in the sequence required by LOGIC or that no mistakes have been made in the spelling or format of the QUESTION card.

E640 The ROWSIZE specified by the teacher is too large for the area provided in LOGIC so that it is not possible to put any text on the screen. This will be the case of an area of four rows, say, is provided by the LOGIC and the ROWSIZE specified by DATA is six rows. The LOGIC should be checked to see what size of area is provided and the teacher should adjust his ROWSIZE accordingly. Also, this condition could arise if the teacher attempts to subscript or superscirpt characters that increase the ROWSIZE outside the limits specified by the logic.

E680 The dictionary character specified for some DATA text is not 0,1,2,3,. As these are the only numbers that are allowed the input card should be checked and changed to produce the desired results.

E710 One of the following illegal conditions have been detected in the ROWSIZE parameter: no parameter at all is present; or the parameter consists of more than two characters; or the parameter is non-numeric; or the parameter is not in the range 1 to 28. The teacher should check to see which rule has been violated and make the required changes according to the specifications in the reference manual.

E710 VAULT expects a QUESTION keyword at this point, to correspond to a POINT in the LOGIC, and one was not found in the DATA input. The teacher should be sure his cards are in the sequence required by LOGIC, or that no mistakes have been made in the spelling or format of the QUESTION card.

E740 More than one $$BLOCK was read in this run of VAULT. As only one BLOCK can be processed at a time, each BLOCK should be separated and processed separately by VAULT. All the Coursewriter II output can then be linked together to form a complete course for the IBM 1500.

E750 More than 99 BLOCK's have been presented to VAULT for processing As 99 is the maximum that can be accommodated, no further processing will be done.

E780 VAULT is unable to process an answer analysis specifying a labelled portion of TEXT. Either the label was not previously defined, that is it was not associated with a particular portion of TEXT or else more than 32 labelled areas were defined for a particular PROBLEM and after the thirty-second label no more were processed by VAULT. The teacher should check the section in the reference manual on Light Pen Responses for a detailed explanation on the correct use of labelled TEXT.

E899 More than 5000 Coursewriter II records have been produced by VAULT and therefore, processing has stopped. This is a restriction of the current version of VAULT being used.

APPENDIX F

Sample VAULT Program

```
STMT.                          V A U L T
 NO.          *** L O G I C   D I V I S I O N ***

    1       $$LOGIC
    2       $$BLOCK
    3       $$UNFECS          NO

    4       $$LESSON

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

    5       $$PROBLEM               ONE
    6       DISPLAY

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

    7       $$PROBLEM               TWO
    8       QUESTION
    9       IF    WRONG
   10       THEN STOP

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

   11       $$PROBLEM               THREE
   12       DISPLAY                              5,10,2,30
   13       DISPLAY   INSERT                     8,10,2,30
   14       DISPLAY   INSERT                    11,10,2,30
   15       DISPLAY   INSERT                    14,10,2,30
   16       POINT

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

   17       $$PROBLEM               FOUR
   18       DISPLAY
   19       $$ENDLOGIC

          * * * E N D   V A U L T   L O G I C   D I V I S I O N * * *
```

```
STMT.                          VAULT
NO.              *** DATA   DIVISION ***

  1       $$DATA
  2       $$COURSENAME          GENE
  3       $$LOGPK               110
  4       $$AUTHOR              A013
  5       $$STARTBLOCK          46
  6       $$BLOCK          ANOVA




  7       $$LESSON         INTRO

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

  8       $$PROBLEM        AN1
  9       $$HYPHEN     10
 10       TEXT: ¬*
 11       ******              <ORGANIZATION OF VAULT ¬$
 12            <MANY FEATURES OF <VAULT> WERE DESIGNED TO FACILITATE THE PR¬#
 13       OGRAMMING OF <CAI> COURSES. <H>OWEVER, THE BASIC DESIGN CONCEPT THAT WA
 14       S INCLUDED WAS SEPARATION OF COURSE LOGICAL STRUCTURE OR TEACHING METHO
 15       DOLOGY FROM THE CURRICULAR CONTENT. <T>HIS MOST IMPORTANT FEATURE MADE
 16       POSSIBLE SUCH AIMS AS REDUCED LEARNING TIME FOR THE TEACHER, DECREASED
 17       CODING EFFORT, INCREASE IN PRODUCTION OF <CAI> COURSES AND INCREASED RE
 18       SEARCH ACTIVITIES.¬$
 19            <B>ASICALLY THEN, <VAULT> IS DIVIDED INTO TWO DIVISIONS, <LOGIC> A
 20       ND <DATA>. <I>T IS THE <LOGIC> DIVISION WHICH SETS THE COURSE STRATEGY,
 21        CHOOSES THE COMPUTER UNITS, AND SPECIFIES THE RECORDING OF STUDENT PER
 22       FORMANCE. <T>HE <DATA> DIVISION PRIMARILY SUPPLIES COURSE MATERIAL AND
 23       EXAMINATION MATERIAL TO BE PRESENTED TO THE STUDENT.¬*
 24            <T>HE SEPERATION, IF IT WAS TO BE EFFECTIVE, REQUIRED SIMILARITIES
 25        IN THE TWO DIVISIONS. <F>OR THIS REASON, TEACHER-ORIENTED SUBDIVISIONS
 26        WERE CHOSEN.

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

 27       $$PROBLEM
 28       QUESTION:      <I>F YOU WOULD LIKE TO CARRY ON WITH THE ORGANIZATION OF
 29       <VAULT> THEN TYPE <YES>, OTHERWISE TYPE <NO
 30       CORRECT: YES:<YES:<Y>ES                  <L>ETS CARRY ON.
 31       WRONG: NO:<NO:<N>'O
 32       UNREC: <T>YPE ONLY <YES> OR <NO
```

```
                                                   ....................
..................

        $$PROBLEM
        TEXT: A: @ <ONE
        TEXT: B: @ <TWO
        TEXT: C: @ <THREE
        TEXT: D: @ <MORE THAN THREE
        QUESTION: <H>OW MANY DIVISIONS DOES <VAULT> CONTAIN?
        CORRECT: B
        WRONG: A:C:D  NO THERE ARE TWO DIVISIONS--TRY <TWO
        $$ENDATA
```

```
        * * * END VAULT DATA DIVISION * * *
```

COURSEWRITER OUTPUT

```
CRDASY
GENF 04600110 INGENE        A013
460000       ANOVA                                                          46000001
        LR  460000¬/RR1                                                     46000002
        LD  B0¬/B5                                                          46000003
        PR                                                                  46000004
        LD  00¬/C03                                                         46000005
        LD  00¬/C06                                                         46000006
        LD  00¬/C11                                                         46000007
        LD  00¬/C26                                                         46000008
        LD  00¬/C27                                                         46000009
460100       INTRO                                                          46000010
        PR                                                                  46000011
        LD  00¬/C02                                                         46000012
        LD  00¬/C05                                                         46000013
        LD  00¬/C10                                                         46000014
        LD  00¬/C25                                                         46000015
        LD  00¬/C30                                                         46000016
460101       AN1          ONE.                                              46000017
        PRR                                                                 46000018
        FN  SF¬/9¬/RR1¬/RR2                                                 46000019
        LR  460101¬/RR1                                                     46000020
        LD  00¬/C01                                                         46000021
        LD  00¬/C09                                                         46000022
        LD  00¬/C13                                                         46000023
        DE  00¬/32                                                          46000024
        DT  12,00¬/02,12¬/40,00¬/           <ORGANIZATION OF VAULT          46000025
        DT  30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--             46000026
        PAE                                                                 46000027
        DE  30¬/02                                                          46000028
        DE  00¬/32                                                          46000029
        DT  00,00¬/02,00¬/40,00¬/     <MANY FEATURES OF VAULT> WERE         46000030
        DT  02,00¬/02,02¬/40,00¬/DESIGNED TO FACILITATE THE PROGRAMMING     46000031
        DT  04,00¬/02,04¬/40,00¬/OF <CAI> COURSES. <H>OWEVER, THE BASIC     46000032
        DT  06,00¬/02,06¬/40,00¬/DESIGN CONCEPT THAT WAS INCLUDED WAS       46000033
        DT  08,00¬/02,08¬/40,00¬/SEPARATION OF COURSE LOGICAL STRUCTURE     46000034
        DT  10,00¬/02,10¬/40,00¬/OR TEACHING METHODOLOGY FROM THE           46000035
        DT  12,00¬/02,12¬/40,00¬/CURRICULAR CONTENT. <T>HIS MOST IMPORTANX46000036
T                                                                           46000037
        DT  14,00¬/02,14¬/40,00¬/FEATURE MADE POSSIBLE SUCH AIMS AS         46000038
        DT  16,00¬/02,16¬/40,00¬/REDUCED LEARNING TIME FOR THE TEACHER,     46000039
        DT  18,00¬/02,18¬/40,00¬/DECREASED CODING EFFCRT, INCREASE IN       46000040
        DT  20,00¬/02,20¬/40,00¬/PRODUCTION OF <CAI> COURSES AND INCREASEX46000041
D                                                                           46000042
        DT  22,00¬/02,22¬/40,00¬/RESEARCH ACTIVITIES.                       46000043
        DT  30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--             46000044
        PAE                                                                 46000045
        DE  30¬/02                                                          46000046
        DE  00¬/32                                                          46000047
        DT  00,00¬/02,00¬/40,00¬/     <B>ASICALLY THEN, <VAULT> IS DIVIDEX46000048
```

```
                                                                         46000049
D
        DT   02,00¬/02,02¬/40,00¬/INTO TWO DIVISIONS, <LOGIC> AND <DATA>. X46000050
                                                                          46000051
<I>T
        DT   04,00¬/02,04¬/40,00¬/IS THE <LOGIC> DIVISION WHICH SETS THE  46000052
        DT   06,00¬/02,06¬/40,00¬/COURSE STRATEGY, CHOOSES THE COMPUTER   46000053
        DT   08,00¬/02,08¬/40,00¬/UNITS, AND SPECIFIES THE RECORDING OF   46000054
        DT   10,00¬/02,10¬/40,00¬/STUDENT PERFORMANCE. <T>HE <DATA> DIVISIX46000055
                                                                          46000056
ON
        DT   12,00¬/02,12¬/40,00¬/PRIMARILY SUPPLIES COURSE MATERIAL AND  46000057
        DT   14,00¬/02,14¬/40,00¬/EXAMINATION MATERIAL TO BE PRESENTED TO 46000058
        DT   16,00¬/02,16¬/40,00¬/THE STUDENT.                            46000059
        DT   18,00¬/02,18¬/40,00¬/     <T>HE SEPERATION, IF IT WAS TO BE  46000060
        DT   20,00¬/02,20¬/40,00¬/EFFECTIVE, REQUIRED SIMILARITIES IN THE 46000061
        DT   22,00¬/02,22¬/40,0C¬/TWO DIVISIONS. <F>CR THIS REASON,       46000062
        DT   24,00¬/02,24¬/40,00¬/TEACHER-ORIENTED SUBDIVISIONS WERE      46000063
        DT   26,00¬/02,26¬/40,Q0¬/CHOSEN.                                 46000064
        DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000065
        PAE                                                               46000066
        DE   30¬/02                                                       46000067
460102              TWO                                                   46000068
        PR                                                                46000069
        FN   SF¬/9¬/RR1¬/RR2                                              46000070
        LR   460102¬/RR1                                                  46000071
        LD   00¬/C01                                                      46000072
        LD   00¬/C0°                                                      46000073
        LD   00¬/C13                                                      46000074
        DE   00¬/32                                                       46000075
        DT   06,00¬/02,06¬/40,00¬/     <I>F YOU WOULD LIKE TO CARRY ON WITHX46000076
                                                                          46000077
        DT   08,00¬/02,08¬/40,00¬/THE ORGANIZATION OF <VAULT> THEN TYPE <YX46000078
                                                                          46000079
ES>,
        DT   10,00¬/02,10¬/40,C0¬/OTHERWISE TYPE <NO                      46000080
        EP   17,00¬/04,¬/40,00¬/0900¬/¬/460102                            46000081
        FN   ED¬/¬/,D,8¬/ ¬/                                              46000082
        NX                                                                46000083
        AD   1¬/C01                                                       46000084
        AD   1¬/C02                                                       46000085
        AD   1¬/C03                                                       46000086
        AD   1¬/C04                                                       46000087
        DT   22,00¬/2,¬/40,00¬/     <I> HAVE NOT RECEIVED YOUR ANSWER.    46000088
        BR   0102¬/C1¬/L¬/03                                              46000089
        DT   24,00¬/2,¬/40,00¬/                <L>ET'S GO BACK.           46000090
        DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000091
        PAE                                                               46000092
        DE   22¬/06                                                       46000093
        DE   30¬/02                                                       46000094
        BR   460101                                                       46000095
        AA   <8                                                           46000096
        AD   1¬/C25                                                       46000097
        AD   1¬/C26                                                       46000098
        AD   1¬/C27                                                       46000099
        CA   YES¬/C1                                                      46000100
        CB   <YES¬/C2                                                     46000101
        CB   <Y>ES¬/C3                                                    46000102
        AD   1¬/C5                                                        46000103
        AD   1¬/C6                                                        46000104
        AD   1¬/C7                                                        46000105
        DT   22,00¬/02,22¬/40,00¬/             <L>ETS CARRY ON.           46000106
        DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000107
        PAE                                                               46000108
```

```
      DE  22¬/06                                                        46000109
      DE  30¬/02                                                        46000110
      BR  460103                                                        46000111
      WA  NO¬/W1                                                        46000112
      WB  <NO¬/W2                                                       46000113
      WB  <N>O¬/W3                                                      46000114
      AD  1¬/C09                                                        46000115
      AD  1¬/C10                                                        46000116
      AD  1¬/C11                                                        46000117
      AD  1¬/C12                                                        46000118
      BR  99999                                                         46000119
      UN  U1                                                            46000120
      DT  22,00¬/02,22¬/40,00¬/<T>YPE ONLY <YES> OR <NO                 46000121
      DT  30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--           46000122
      PAE                                                               46000123
      DE  22¬/06                                                        46000124
      DE  30¬/02                                                        46000125
      BR  RE                                                            46000126
0102                                                                    46000127
      EA                                                                46000128
      DT  24,00¬/2,¬/40,00¬/            <H>AVE ANOTHER TRY.             46000129
      DT  30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--           46000130
      PAE                                                               46000131
      DE  22¬/06                                                        46000132
      DE  30¬/02                                                        46000133
      BR  RE                                                            46000134
460103                THREE                                             46000135
      PR                                                                46000136
      FN  SF¬/9¬/RR1¬/RR2                                               46000137
      LR  460103¬/RR1                                                   46000138
      LD  00¬/C01                                                       46000139
      LD  00¬/C09                                                       46000140
      LD  00¬/C13                                                       46000141
      DE  00¬/32                                                        46000142
      DT  05,10¬/02,05¬/30,10¬/@ <ONE                                   46000143
      DTI 08,10¬/02,08¬/30,10¬/@ <TWO                                   46000144
      DTI 11,10¬/02,11¬/30,10¬/@ <THREE                                 46000145
      DTI 14,10¬/02,14¬/30,10¬/@ <MORE THAN THREE                       46000146
      DT  00,00¬/02,00¬/40,00¬/<H>CW MANY DIVISIONS DOES <VAULT> CONTAIX46000147
N?                                                                      46000148
      EPP 0900¬/460103                                                  46000149
      NX                                                                46000150
      AD  1¬/C01                                                        46000151
      AD  1¬/C02                                                        46000152
      AD  1¬/C03                                                        46000153
      AD  1¬/C04                                                        46000154
      DT  22,00¬/2,¬/40,00¬/         <P>LEASE RESPOND QUICKLY.          46000155
      BR  0103¬/C1¬/L¬/03                                               46000156
      DT  24,00¬/2,¬/40,00¬/ <L>ET'S REEXAMINE THE PREVIOUS MATERIAL.   46000157
      DT  30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAP TO CONTINUE>--           46000158
      PAE                                                               46000159
      DE  22¬/06                                                        46000160
      DE  30¬/02                                                        46000161
      BR  460101                                                        46000162
      AA  <8                                                            46000163
      AD  1¬/C25                                                        46000164
      AD  1¬/C26                                                        46000165
      AD  1¬/C27                                                        46000166
      CAP 02,08,30,10¬/C1                                               46000167
      AD  1¬/C5                                                         46000168
```

```
          AD   1¬/C6                                                        46000169
          AD   1¬/C7                                                        46000170
          DT   22,00¬/2,¬/40,00¬/                  <CORRECT>.               46000171
          DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUF>--          46000172
          PAE                                                               46000173
          DE   22¬/06                                                       46000174
          DE   30¬/02                                                       46000175
          BR   460104                                                       46000176
          WAP  02,05,30,10¬/W1                                              46000177
          WBP  02,11,30,10¬/W2                                              46000178
          WBP  02,14,30,10¬/W3                                              46000179
          AD   1¬/C09                                                       46000180
          AD   1¬/C10                                                       46000181
          AD   1¬/C11                                                       46000182
          AD   1¬/C12                                                       46000183
          DT   22,00¬/02,22¬/40,00¬/ NO THERF ARE TWO DIVISIONS--TRY <TWO   46000184
          DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000185
          PAE                                                               46000186
          DE   22¬/06                                                       46000187
          DE   30¬/02                                                       46000188
          BR   RE¬/C9¬/L¬/03                                                46000189
          DT   24,00¬/2,¬/40,00¬/        <L>ET'S REDO A FEW PROBLEMS.       46000190
          DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000191
          PAE                                                               46000192
          DE   22¬/06                                                       46000193
          DE   30¬/02                                                       46000194
          BR   460101                                                       46000195
          UN   U1                                                           46000196
          DT   22,00¬/2,¬/40,00¬/            <Y>OU DID NOT GET IT.          46000197
          DT   24,00¬/2,¬/40,00¬/             <H>AVE ANOTHER TRY.           46000198
          DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000199
          PAE                                                               46000200
          DE   22¬/06                                                       46000201
          DE   30¬/02                                                       46000202
          BR   RE                                                           46000203
0103                                                                        46000204
          EA                                                                46000205
          DT   24,00¬/2,¬/40,00¬/             <H>AVE ANOTHER TRY.           46000206
          DT   30,4¬/2,30¬/40,0¬/--<PRESS SPACE BAR TO CONTINUE>--          46000207
          PAE                                                               46000208
          DE   22¬/06                                                       46000209
          DE   30¬/02                                                       46000210
          BR   RE                                                           46000211
460104                                                                      46000212
99999                                                                       46000213
          EN                                                                46000214
*END                                                                        46000215
```