8-1981

A Computer-Assisted Instructional System and its Application to a Course on Computer Structures and Hardware

Tahir Mufti

Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/1817

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
A COMPUTER-ASSISTED INSTRUCTIONAL SYSTEM
AND ITS APPLICATION TO A COURSE ON
COMPUTER STRUCTURES AND HARDWARE

by

Tahir Mufti

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
August 1981
A COMPUTER-ASSISTED INSTRUCTIONAL SYSTEM
AND ITS APPLICATION TO A COURSE ON
COMPUTER STRUCTURES AND HARDWARE

Tahir Mufti, M.S.
Western Michigan University, 1981

Computer-Assisted Instruction is a powerful tool which is fast becoming an integral part of the instructional system in schools.

The objective of this study is to review the techniques used to provide Computer-Assisted Instruction and to present such an instructional system called LEARN which is general enough to be applied towards the teaching of any subject. Also included are notes on the application of the LEARN system to a subject entitled "Computer Structures and Hardware".
ACKNOWLEDGEMENTS

I am greatly indebted to my Thesis Advisor, Dr. Dionysios Kountanis, for his advice and encouragement during the design and implementation of the LEARN Computer-Assisted Instruction system.

Tahir Mufti
INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.

2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame. If copyrighted materials were deleted you will find a target note listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in 'sectioning' the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.

University
Microfilms
International
300 N. ZEEB RD., ANN ARBOR, MI 48106

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
MUFTI, TAHIR
A COMPUTER-ASSISTED INSTRUCTIONAL SYSTEM AND ITS APPLICATION TO A COURSE ON COMPUTER STRUCTURES AND HARDWARE.
WESTERN MICHIGAN UNIVERSITY, M.S., 1981
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark √.

1. Glossy photographs or pages ______
2. Colored illustrations, paper or print ______
3. Photographs with dark background ______
4. Illustrations are poor copy ______
5. Pages with black marks, not original copy ______
6. Print shows through as there is text on both sides of page ______
7. Indistinct, broken or small print on several pages √
8. Print exceeds margin requirements ______
9. Tightly bound copy with print lost in spine ______
10. Computer printout pages with indistinct print √
11. Page(s) ___________ lacking when material received, and not available from school or author.
12. Page(s) ___________ seem to be missing in numbering only as text follows.
13. Two pages numbered ___________. Text follows.
14. Curling and wrinkled pages ______
15. Other __________________________________________________________________________

University
Microfilms
International

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# TABLE OF CONTENTS

**ACKNOWLEDGEMENTS**

**LIST OF TABLES**

**LIST OF FIGURES**

Chapter

I. INTRODUCTION

II. REVIEW OF CAI TECHNIQUES

  2.1 Computer-Managed Instruction

  2.2 Computer-Assisted Instruction

    2.2.1 The drill-and-practice technique

    2.2.2 The tutorial technique

    2.2.3 Selective Teaching Systems

    2.2.4 Generative Teaching Systems

    2.2.5 Self-paced study

III. COMPUTER MODEL OF A TUTOR

  3.1 Analysis of the tutor model

    3.1.1 Displaying text

    3.1.2 The student

    3.1.3 Accepting the response

    3.1.4 Evaluation of the response

    3.1.5 Analysis of the response

    3.1.6 Selection of remedial material

    3.1.7 Selection of text

III
3.1.8 Generation of problems ........................................26
3.2 Design of a lesson .................................................27

IV. DESIGN OF THE SYSTEM ..............................................32
4.1 An overview of the system ........................................32
4.2 Organization of the system ......................................35
4.3 Files accessed by the system ..................................37
  4.3.1 The TEXTFILE ..............................................37
  4.3.2 The DIRECTORY ...........................................44
  4.3.3 The RECORD-FILE .........................................47
4.4 The Teaching System ............................................49
  4.4.1 The Initialization Module .................................49
  4.4.2 The Query Module .........................................54
  4.4.3 The Directory Module ....................................55
    4.4.3.1 Organization of the internal directory ........56
  4.4.4 Teach Module ..............................................61
    4.4.4.1 The :TEXT instruction ...............................65
    4.4.4.2 The :RIGHT instruction ..............................65
    4.4.4.3 The :WRONG instruction ..............................66
    4.4.4.4 The :MATCH instruction ..............................66
    4.4.4.5 The :REPLY instruction ..............................67
    4.4.4.6 The :JUMP instruction ...............................67
    4.4.4.7 The :INSERT instruction .............................68
    4.4.4.8 The :PAUSE instruction ...............................69
    4.4.4.9 The :Call instruction ................................69
    4.4.4.10 The :RUN instruction ...............................70
    4.4.4.11 The /END instruction ...............................70
4.5 The Editor

V. FUTURE EXPANSION AND MAINTENANCE

5.1 Possible expansions

5.1.1 Use of artificial intelligence

5.1.2 Interfacing TEXTFILE and user routines

5.1.3 Query and directory-search techniques

5.1.4 Report generation

5.1.5 Extending the TEXTFILE instruction set

5.1.6 The Editor

5.1.7 Graphic capabilities

5.2 Limitations of the system

5.3 Maintenance of the system

APPENDIX A - INSTRUCTOR-USER'S MANUAL

APPENDIX B - STUDENT-USER'S MANUAL

APPENDIX C - NOTES ON APPLICATION TO COMPUTER STRUCTURES

APPENDIX D - SYSTEM DIRECTORY

APPENDIX E - SAMPLE OF TEXTFILE

APPENDIX F - SAMPLE RUNS

BIBLIOGRAPHY
<table>
<thead>
<tr>
<th>TABLE NO.</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>62</td>
</tr>
</tbody>
</table>

SET OF TEXTFILE INSTRUCTIONS
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE NO.</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>COMPUTER MODEL OF A TUTOR</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>DESIGN OF A LESSON</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>DESIGN OF A SELECTIVE TEST</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>DESIGN OF A GENERATIVE TEST</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>BLOCK DIAGRAM OF THE LEARN SYSTEM</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>FILES USED BY THE SYSTEM</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>A SAMPLE TEXTFILE</td>
<td>43</td>
</tr>
<tr>
<td>4.4</td>
<td>STRUCTURE OF THE TEXTFILE RECORD</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>STRUCTURE OF THE DIRECTORY-FILE RECORD</td>
<td>45</td>
</tr>
<tr>
<td>4.6</td>
<td>A SAMPLE DIRECTORY-FILE</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>STRUCTURE OF A RECORD-FILE RECORD</td>
<td>48</td>
</tr>
<tr>
<td>4.8</td>
<td>THE TEACHING SYSTEM</td>
<td>50</td>
</tr>
<tr>
<td>4.9</td>
<td>THE QUERY MODULE</td>
<td>51</td>
</tr>
<tr>
<td>4.10</td>
<td>THE DIRECTORY MODULE</td>
<td>52</td>
</tr>
<tr>
<td>4.11</td>
<td>LOGICAL ORGANIZATION OF THE DIRECTORY</td>
<td>53</td>
</tr>
<tr>
<td>4.12</td>
<td>THE DIRECTORY NODE STRUCTURE</td>
<td>58</td>
</tr>
<tr>
<td>4.13</td>
<td>THE TEACH MODULE</td>
<td>64</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

The use of a computer as a tool in teaching is relatively new, but it has been increasing rapidly in the last few years. Many major universities have developed or are developing courses with the computer used for instructional purposes.

As hardware and software costs decrease, it becomes more and more economically feasible to explore this use of the computer. Another reason for increased activity in computerized instruction is the advent of mini and micro-computers. It has become possible for schools to invest in such small systems which is something they could not have done before.

Other factors which have increased the potential of computers in teaching are the availability of time-sharing which permits access to a remote computer, and the simplification of programming which allows a user to make effective use of the computer without an extensive knowledge of computer hardware.
In education the computer has been used for many different purposes, varying from vocational guidance in curriculum, scheduling of classes, administrative record keeping, test generation and analysis, to completely self-paced study.

As a result of research and experimentation at a number of centers, courses have been developed in various disciplines including mathematics, physics, chemistry, computer sciences, engineering, foreign language training, psychology, statistics, economics, management science, medical and biological sciences.

Interest in computer-assisted instruction has given birth to special purpose languages known as "CAI authoring languages". These languages allow a teacher with little or no knowledge of computer programming to set up and utilize CAI lessons. The original language to attain wide usage was COURSEWRITER, developed by IBM. Since then various other systems have been developed. PLANET, PILOT, DITRAN, PLATO, GNOSIS are but a few of these.

In this thesis the design and implementation of a computer-assisted authoring language called LEARN is presented. Also presented is the application of the
system to a course designed to teach the basics of computer structures and hardware. The discussion on the LEARN CAI system itself has been kept independent of the application. The design of the system is presented in chapter 3 whereas notes on the application are included in the appendices.

The objective of this thesis is not to demonstrate the superiority or effectiveness of this system as compared to other existing systems, rather it is to present a study of the techniques used in the design and implementation of such a system, and to discuss the factors which influence the effectiveness and usefulness of instruction given through such systems.

The system itself is written in PASCAL and has been implemented on a Digital Equipment Corporation PDP-10 computer at Western Michigan University.

Successful instruction by a computer involves more than simply providing a set of computer programs. The design of a CAI system depends greatly on a study of the philosophies of teaching and learning (Arms, 1977). The design of the system described in this thesis has definitely been influenced by research done in related fields. A discussion and review of related research is
included in chapter 2.

Before the design of the system itself is given, a computer-model of a tutor along with the ideas which have affected the design of the system are described in chapter 3. This includes a discussion on the techniques used in writing instructional text and in generating test and quizzes, which have been used in the application of the system to the teaching of computer structures and hardware.

Chapter 4 demonstrates the logical design of the system and the techniques used in the implementation. The four main modules of the system and their organization is described in detail: (1) the logic of the query system and the query language, (2) the structure of the system directory and the directory search techniques, (3) the teaching module and the structure of the instructional data-base and (4) the system's editing facilities which may be used to create and maintain the instructional data-base. Specific details of the implementation are described where necessary.
Chapter 5 describes the capabilities and limitations of the system, especially the storage requirements of the system. It also includes notes on future expansions and possible modifications to the system.

The appendices include detailed user-manuals for the instructor who operates the system and the student who uses the system. Also included is an outline of the material contained in the application of the system to computer structures and hardware. The actual text and the system source-program are not included due to enormous size.
There are various terms used to describe the instructional use of a computer: CAI (computer-assisted instruction or computer-aided instruction), CAL (computer-aided learning), CBI (computer-based instruction), CAT (computer-assisted testing), CMI (computer-managed instruction) etc., the most popular ones being CAI (Computer-Assisted Instruction) and CMI (Computer-Managed Instruction). It is possible to classify most computerized instructional systems into either of these categories, however, this classification is not very distinct, there being obvious overlaps between the categories.

2.1 Computer-Managed Instruction

CMI refers to a system in which the computer is used as a monitor and as an information system for designing instruction generally produced in other ways (Kieren, 1973). A CMI system would generally include a computer-assisted testing system in which tests are generated and scored by a computer, plus computer-managed
reporting which would be used to analyse the test results and generate reports for the teacher. The teaching itself however is done more or less in conventional ways. Most CMI systems are operated off-line in a batch mode and the student does not actually interact with the computer.

2.2 Computer-Assisted Instruction

The term CAI on the other hand refers to a system in which the student actually interacts with a computer. The interaction may be of the drill-and-practice type, or it may be tutorial (Kieren, 1973).

2.2.1 The drill-and-practice technique

The drill-and-practice technique attempts to develop the student's skills under controlled learning conditions, permitting him to practice that skill under a wide variety of learning situations (Molnar, 1973). Usually the drills are set up as a sequence of multiple choice questions arranged in "frames". The student is allowed to select between a limited number of possible responses; the student's responses are evaluated and used to alter the logical sequence of the lesson.
accordingly. In case of a correct answer, the system branches off to the next logically-higher frame, while in case of a wrong answer, remedial material is presented.

2.2.2 The tutorial technique

In the tutorial technique, the student receives instruction on new material directly from the computer. The student may carry on a question-answer dialog with the computer, having considerable freedom and choice in selecting the text to be studied. The text is usually arranged in levels of increasing difficulty. The student's responses are scored and his progress evaluated and analysed at each step. Depending on the results of this analysis, the system adjusts the logical sequence of the lesson, branching off to the appropriate levels. The system provides the student with helpful hints for problem-solving or the student may query the computer for reasonable approaches to a particular problem.

A version of the tutorial technique is what is known as the inquiry system. Such systems allow greater freedom to the student. On being presented by a problem, it is left to the student to inquire and probe the system for possible paths to the solution. The student is given
one of a number of directions. This approach is quite different from the conventional tutorial approach where the system presents specific questions and expects a specific answer.

As this thesis is concerned with the tutorial aspects of CAI, related research and the various techniques used to design and implement such systems will be reviewed.
2.2.3 Selective teaching systems

Computer-assisted instruction systems may be classified as being either "selective" or "generative" systems.

Selective teaching systems usually consist of a large data-base of text which is randomly accessible. The system operates in a logical sequence of pre-programmed questions, the student's response being compared to a pre-programmed list of possible responses. The system then displays an appropriate message depending on the correctness of the response. Most of the earlier work done in CAI was of selective nature, composed of frames presenting a specific question and a list of possible responses from which the student is asked to choose the correct one. The various CAI authoring languages described earlier, in their basic form, make it easy to construct such systems.

Selective systems have several practical limitations: The on-line storage capacity of the computer has to be fairly large to be able to store the entire data-base of text. Moreover, the teacher has to pre-program the system with all the possible responses.
that the student might enter, which is a tedious if not impossible task.

2.2.4 Generative teaching systems

The organization of generative computer-assisted instruction systems is different from selective systems. Instead of being based on a data-base of text, these systems are based on a set of powerful algorithms which are capable of generating problems, answers and evaluation standards. Such systems are appropriate to the teaching of mathematically oriented subjects like physical sciences, chemistry and logic in which there is a set of defined relations on which the algorithms can be constructed. As the questions and the possible responses are generated by the system itself it is no longer necessary for the teacher to go through the process of anticipating and pre-programming such responses. The recent research and development in artificial intelligence has made it possible to build systems which impose less restrictions on the dialog carried out between the computer and the student, allowing for a more natural dialog.
Although most CAI authoring languages may be used to set up generative lessons by themselves, or by interaction with external user-written routines, most researchers have shown a preference to use other computer languages independent of authoring languages. Even if authoring languages are used for this purpose, the only important function they perform is to log the student's progress and to prepare records for the teacher, depending on the extent of interaction allowed between the authoring system and the user-written routines.

Apart from the techniques which are used to store the instructional material, or to generate text and/or tests, there are various other factors which affect the effectiveness of a CAI system, the most important ones being the extent of self-pacing allowed to the student and the degree of control the student has over the text to be studied.

2.2.5 Self-paced study

Studies in CAI have resulted in debates on the important issue of self-paced study. Most studies have shown that maximum student achievement was provided for when the time allowed for learning could vary (Carrol,
However, as Briggs (1968) has pointed out, neither pushing the student through the program faster, nor self-pacing by the student is the ultimate solution to providing better learning. It is necessary to diagnose the student's progress and control the learning process accordingly. The student must be directed through a logical sequence of material which can build up his skills and knowledge on firm grounds. The importance of the contents of the instructional material is also stressed by Bork (1975):

The test of all learning is with students and the value of teaching materials, computer or otherwise, depends upon whether or not they lead to some type of learning for the students. (pp. 4-11).

This shows that the logical organization of the teaching system and the means of evaluating the student's progress are as important as is the instructional text. Moreover, the system must be capable of interpreting the student's response and providing meaningful feedback.

Barr, Beard, Atkinson and Lorton (1974) have worked on a system to teach programming languages. They note that:

A major goal of the research project is to increase the sophistication with which the instructional program monitors the student's
work and responds to him with appropriate hints and prompts. (p. 243).

They stress the use of algorithms to check the student's progress whenever possible, and consider it practical to develop verification techniques that are general in nature and applicable to other technical areas.

Courses in technical areas (logic and integral calculus) have been developed at the Institute of Mathematics in Social Sciences at Stanford University (Goldberg, 1973 and Kimball, 1977) which are capable of dealing in a sophisticated way with both their subject matter and with the student. These courses provide instructive interaction at all times during the problem solving activity by performing operations specified by the student, evaluating the effect of such operations and on request suggesting a hint to the solution.

Gagne (1961) of Florida State University has stressed the need for the designer of a CAI system to perform a task analysis in order to identify the learning hierarchy, to sequence the learning "units" properly and insist that lower level units be mastered before the student is permitted to proceed to the next higher level unit.
Evans (1977) advocates this in his discussion on a project which uses the "answer-until-correct" procedure, providing the student with feedback on why the previous answer was incorrect, and prompting him to try again.

Although most researchers agree on the point of mastering a unit before being permitted to proceed to the next level, they also recognise that students must be allowed to retake alternate tests. It is a general attitude to allow retakes without any restrictions, using this as a means of developing the student's skills until he masters the subject on hand completely. This is, after all, the objective of the teaching system.

Another question which is often debated is whether to allow a student to control the sequence of the material which is to be taught, or should this be pre-determined by the system designer?

Grubb (1971) has established that by giving complete control to the student no significant gains are achieved. However, this does not mean that the student should not have any say in the learning process. It is possible to construct systems which allow a student to select from a number of "paths" that he may take to reach a common objective. Such a system designed to teach statistics
has been demonstrated by Grubb. At each level the student is allowed to choose between sub-sections of the topic, or if he feels he has mastered the topic, to move back to a higher level and proceed in a different direction. Grubb describes this approach as a network of concepts through which the learner is free to travel.

Compared with conventional means of teaching, CAI generally results in better achievements. In one case it has been shown that adults learning computer programming through CAI took five to ten hours to reach the same level of proficiency as students who spent from 24 to 30 hours in traditional instruction (Grubb, 1971).

Various other studies can be quoted to establish the importance of CAI at all levels of learning. Computer simulation of high school physics experiments compared with traditional laboratory experiments showed that CAI students learned more in one-eighth the time (Grubb, 1971).

Programs have been developed at Stanford University (Suppes and Morningstar, 1972) to teach mathematics to elementary school students. Results have shown significant improvements in learning rates.
Gosman, Launder, Lockwood and Reese (1976) have developed a system to teach fluid mechanics and heat transfer. They quote students' perception of the subject as: "It makes things more visual in a sense" (p. 111), and "It gives you a sense of physical reality". (p. 113).

One thing which must be realized is that it is not the equipment or the gadgetary used to construct the system which determines its usefulness, it is the thought and logic behind the design of the system which can result in better achievements.
CHAPTER III

COMPUTER MODEL OF A TUTOR

A well designed CAI lesson is a simulation of normal teacher-student interactions. It is not the program but the program designer who is the real teacher in a CAI system.

The human teacher functions in ways which are complex and are not based on well defined rules. This complicates the process of analysing and simulating this process.

Models have been constructed to describe the process of teaching which are mostly descriptive and procedural rather than being structural in nature. Such a model is shown in figure 3.1, which is a very simplified model of the human tutor. This demonstrates the closed-loop interactive process carried out between the teacher and the student.
Figure 3.1. Computer Model of a Tutor
3.1 Analysis of the tutor model

The model operates in a closed-loop feedback system. A question or problem is presented by the tutor. The student perceives the meaning of the question and in the context of his previous experience processes it. He then decides on an appropriate response and conveys it to the tutor.

The conveying of a response from the student to the tutor is perhaps the most limiting link in the computer model. Computers cannot accept strings of verbal or symbolic input and process them in the same way that a human being can process very complex patterns. This imposes strict restrictions on the format of the response which can be conveyed by the student. Recent developments in artificial intelligence have resulted in systems which are capable of processing subsets of natural languages, however, even such systems are restricted in the patterns they can recognize due to that fact that they can not simulate the human mind in recognizing patterns, rather they depend on "tricks" to recognize and process patterns.
Once the student's response has been acquired it must be evaluated. At the simplest "selective" level this would be a simple character-by-character comparison with a list of pre-programmed responses. In a generative system, however, the student's response may be fitted into the system concept structure although the response may not be in the most desirable form.

If it is determined that an answer to a question is correct, the tutor moves on to the next step in the curriculum. If an error is made, however, the situation is much more complicated. It is necessary to diagnose the response and present appropriate remedial material. As can be seen from the model, this process of providing remedial material is recursive, going deeper and deeper until the student achieves an acceptable level of understanding.

After remedial material has been presented, the tutor returns to the main stream of the curriculum and again tests the student to see if the trouble has been corrected.

The feedback loop is closed by presenting new text or generating a new problem for the student.
The tutor model illustrated in figure 3.1 is the basis of the LEARN CAI system. A fairly detailed discussion on the different blocks comprising the model is given below. The ideas and techniques discussed reflect the design and implementation of the LEARN system.

3.1.1 Displaying text

The actual process of displaying text to the student is not complicated at all. What is important is the source of the text. The text may be simply retrieved from a data-base of information or it may be generated by the system. These two approaches are discussed later in the appropriate sections. At this time it will be sufficient to note that when a question or problem is displayed, the student may be offered the following options for the response format:

1. The question may be displayed as a "multiple-choice question" asking the student to choose from a list of possible responses, or

2. The question may ask for a verbal response, allowing the student to select appropriate words to compose the response.
3.1.2 The student

The student accepts the question and decides on a response. Once this has been done, the student enters his answer using the options available to him. In case of a multiple-choice question he may enter a number or a letter as specified in the question, or in case of a verbal response being required, type in a response composed of meaningful words.

3.1.3 Accepting the response

Once the student types in a response, the system has to change it to a standard form. In case of a response to a multiple-response question this is an easy matter, however in case of a verbal response it is necessary to employ complex algorithms to format the response so that it may be processed. In most cases the system looks for "keywords" and isolates them so that they may be used to evaluate the correctness of the response.
3.1.4 Evaluation of the response

The student's response is then evaluated. This evaluation may be:

(i) Of the simple character-by-character type, in which case the student's response is compared to a pre-programmed list of possible responses.

(ii) The system may look for certain keywords in the response and interpret them free of their context.

(iii) More complex techniques might be used to interpret the keywords in the light of their context.

The LEARN system provides for all of these options, however, the capabilities to search for keywords in their context is very restricted and in no way approaches the capabilities of complicated natural-language processors.

3.1.5 Analysis of the response

Once it has been determined that the student's response was incorrect, it is necessary to decide on the remedial material which must be presented to the student. In most cases the present status of the student including his score and the current topic being studied may be enough to classify the remedial material required. In
other cases the system may carry out a short question-answer dialog with the student to pin-point the problem. The student's previous record may also be an important figure in the analysis.

3.1.6 Selection of remedial material

Once the student's problem area has been determined, it is necessary to present remedial material for it. This material may be part of the material which the student previously studied or it may consist of some additional material. In either case, the techniques described in the following two sections apply.

3.1.7 Selection of new text

Whether the student's response was correct or incorrect, it will be necessary to present new material. This material may come from two possible sources. (1) It may be generated by the system or (2) it may be retrieved from a data-base of information. The latter alternative is used in case of definitions and basic concepts or descriptions which are impossible to generate through algorithms. This results in a simple case of retrieving the necessary information and displaying it. The only
complication is in the design of the material itself, which is independent of the system.

3.1.8 Generation of problems

Certain material is based on a well defined set of relations and therefore can be generated through the use of appropriate algorithms. This includes drills on mathematically based concepts and the generation of related problems. In the most simplest form, algorithms may be used to generate random numbers and "plug" them into a certain problem. The algorithm would also be responsible for determining the correctness of the response entered by the student. Such generative systems are much more flexible because it is not necessary for the designer to pre-program all the possible responses to a question. However, in some cases it is not possible to employ generative techniques, and therefore it is necessary to store the questions and the possible responses as part of the text.

Once a problem or text has been selected it is displayed to the student, thus closing the teacher-student feedback loop.
3.2 Design of a lesson

The flowchart in figure 3.2 demonstrates the logical structure of a lesson. The student is presented with new material on a certain subject. This is followed by a quiz on the subject. The new material would normally be retrieved from the data-base of information, whereas the quizzes might be generated by the system, or they might also be retrieved from the data-base. Figures 3.3 and 3.4 show the logical design of selective and generative drills respectively. The student's performance in the quiz determines whether to proceed to the next topic or to present remedial material. If it is necessary to present remedial material then the student is again quizzed on this remedial material. This starts a recursive process which continues until the student has achieved an acceptable level of understanding.

The most important difference between selective and generative drills is the way in which the student's response is evaluated. In a selective design, the system designer would have to pre-program all the possible responses to a question, whereas in a generative design the student's response is fed as input to an algorithm which determines if it is correct or incorrect. This
frees the system designer from having to foresee all possible responses to a certain problem. In addition to this a generative system is capable of generating an unlimited number of questions and thus uses much less storage space compared to a selective system in which all problems and their possible responses would have to be stored separately.
START LESSON

Select appropriate material and display

Quiz on new material

Acceptable comprehension?

Yes

Move to next higher level

No

Lesson finished?

Yes

STOP

No

Move to lower level

Figure 3.2. Design of a lesson.
Figure 3.3. Structure of a selective test
Figure 3.4. Structure of a generative test
CHAPTER IV

DESIGN OF THE LEARN SYSTEM

4.1 An overview of the system.

From the teacher's point of view the objective of this system is to provide an easy means of using a computer to teach and then test students on any selected subject.

To use the system the teacher need not have any knowledge of programming languages. The teacher can set up and maintain a TEXTFILE containing material which is to be used for teaching by using a few simple instructions. The system provides its own text editor program to help with this process.

The teacher can restrict himself to the instructions provided by the system and thus set up lessons which are selective in nature, or he may combine the instructions provided by the system with calls to external procedures and thus set up lessons which are generative.
Figure 4.1. Block structure of the LEARN system

Figure 4.2. Files accessed by the system
From the student's point of view the objective of the system is to provide an easy way of learning about a particular subject and then testing the knowledge thus acquired. The three options available to the student user are:

(i) To use the system command LEARN and retrieve information about a particular topic.

(ii) Use the command EXAMPLES to retrieve examples pertaining to a particular topic.

(iii) Use the command TEST to test his knowledge on a particular topic.

There are no time limits imposed on the student, so that he may progress through the material at his own pace.

The system keeps a record of the student users' activities. The most important statistics being the test scores. These statistics are available to the teacher who is operating the system.
4.2 Organization of the system.

The system is composed of the following two components:

(i) the teaching program and
(ii) the text editor.

(i) The teaching program itself consists of these four modules (figure 4.8):
   a) The INITIALIZATION module
   b) The QUERY module
   c) The DIRECTORY module
   d) The TEACH module

The logical design of these four modules is discussed section 4.4. The actual techniques used to implement them are described where necessary.

The fact that the primary purpose of this system is teaching has undoubtedly been the foremost consideration during the design phases. At the same time, however, it was realized that the system falls into the category of being a data-base and as such, two of the most important design considerations were the structure of the data (text) records and the structure of the system directory. Trade-offs had to be reached in the cases of disk access-time and disk storage space requirements to
provide for greater flexibility and ease of maintaining the system. These factors are discussed in detail in the following pages.

(ii) The text editor allows the teacher to create and edit the TEXTFILE which contains the instructional material. Notes on the use of the EDITOR are included in the INSTRUCTOR'S MANUAL in appendix A.
4.3 Files accessed by the system.

The system uses the following files (figure 4.2):

a) TEXTFILE (The instructional material)
b) DIRECTORY (The system directory)
c) RECORD-FILE (student statistics)
d) TEMP-FILE (temporary file)
e) OLDFILE (temporary file)

The organization of these files is discussed in this section. TEMP-FILE and OLDFILE are scratch-pad files and therefore are not included in the discussion.

4.3.1 The TEXTFILE:

This is a disk file created by the teacher and contains the instructional material. The teacher may use the EDITOR program to create this file.

Details on how to use the EDITOR to create the TEXTFILE are included in the teacher's manual.
The TEXTFILE is organized into variable-size records. There is no practical limit on the size of a record. A record may contain as many lines as necessary. The instructional text for a particular topic may be contained in a single record or may cover several records. Records may be left independent or they may be linked together depending on the logic of the particular lesson.

As mentioned earlier, one of the most important design considerations was the structure of the text records in the TEXTFILE. Obviously it is necessary to provide random access to the text records. However there were two serious problems: first, PASCAL the implementing language does not support random access of files. Second, even if another language which provides random access were used there would still be the problem of not being able to use variable-length records. It is a simple matter to realize that text records cannot be of fixed size. In most other applications dealing with variable sized records it is possible to classify records into a number of sizes and then use techniques to provide random access for these limited number of sizes. In this case, however, it is not possible to do this because no two text records can be expected to be of the same size.
Another two considerations which do not allow for techniques which would require breaking up records into parts and storing them in some fixed format are that the user should be able to create and edit the TEXTFILE using any text editor, and that the TEXTFILE should be in a readable form for easy reference to the material contained in it.

These considerations resulted in the designing of routines to provide random access to variable-length records.

Although this feature of the system is dependent on the DEC-system PDP-10, the dependency has been kept to a minimum to allow for portability of the system.

On the DEC PDP-10 one disk block contains 128 words. Before performing an input from disk, it is possible to send out a "monitor call" to position the disk "read-head" to the beginning of a specified block. After this every input instruction will result in the next 640 characters being read in.

To access the correct block, the LEARN system stores the block number corresponding to the beginning of each record in a table. After a directory search, when it has
been determined which record must be retrieved, the
system looks up the corresponding block, positions the
disk read-head and then starts reading from disk.
Because the TEXTFILE record numbers do not (and in fact, cannot) coincide with the block boundaries, it is
necessary to read through some part of the block to reach
the beginning of the record. This does result in an
access time higher than what would be expected with
fixed-length records, but the trade-off in light of the
advantages gained was considered to be reasonable.

Each record in the TEXTFILE begins with a
RECORD-NUMBER and ends with an END-OF-RECORD mark. The
body of the record may contain one or more of the
following marks:

(i) : n (the record number)
(ii) :TEXT (start of text)
(iii) :RIGHT (possible correct response)
(iv) :WRONG (possible wrong response)
(v) :MATCH (possible response)
(vi) :REPLY (display text if response matches)
(vii) :SELECT n (select and display 1 of n TEXTS)
(viii) :PAUSE (pause while displaying text)
(ix) :JUMP n (retrieve and process record n
immediately)
(x) :INSERT n (retrieve and process record n after current record)

(xi) :CALL n (call an external routine)

(xii) :RUN name (Run a systems program)

(xiii) /END (end of record)

A detailed description of what each of the above mark signifies is given in section 4.4.4.

The only three restrictions which must be observed while creating the TEXTFILE are:

(i) in the TEXTFILE each mark must be preceded by a colon (:) in column 1 of a new line, immediately followed by the mark. The actual text may begin in column eight of the same line, and may continue for any number of lines, always beginning in column eight.

(ii) the record numbers must be assigned in ascending order. However, they need not be consecutive. This allows the user to use a step of 10 (or any other appropriate step) when initially creating the file so that future insertions may be made.

(iii) the first line of the file must contain the record number of the last record. This is used by the system to
check for possible errors in the directory.

- If the system EDITOR is being used then the user need not be concerned with the above three restrictions, the editor program takes care of them.

A sample TEXTFILE is shown in figure 4.3, and the structure of a text record is shown in figure 4.4.
2 RECORDS IN THIS TEXTFILE

: 1
:TEXT HOW MUCH IS THE SUM OF 0010 AND 1100 ?
:RIGHT 1110
:REPLY YES, THAT IS CORRECT.
:WRONG *
:REPLY SEEMS LIKE YOU NEED TO BRUSH UP ON YOUR BINARY ARITHMETIC!
:TEXT AND HOW MUCH IS 01 TIMES 0011 ?
:RIGHT 0011
:JUMP 2
:WRONG *
:REPLY YOU SHOULD LOOK AT "BINARY ARITHMETIC" BEFORE YOU ATTEMPT TO PROCEED.
:JUMP 0
:/END

: 2
:TEXT NOW LETS SEE HOW WELL YOU CAN DIVIDE IN BINARY!
WHEN YOU DIVIDE 01100 BY 10 HOW MUCH DO YOU GET ?
:RIGHT 0110
:/END

Figure 4.3. A sample TEXTFILE

<table>
<thead>
<tr>
<th>Record no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions plus lines of text</td>
</tr>
<tr>
<td>/END</td>
</tr>
</tbody>
</table>

Figure 4.4. Structure of the TEXTFILE record.
4.3.2 The DIRECTORY:

The DIRECTORY file must also be created by the teacher. To process a student query, the system searches the directory for a possible match, therefore all records contained in the TEXTFILE must be accessible either directly or indirectly (linked through other records) through the directory.

The directory is composed of three sub-directories, one for each of:

(i) LEARN
(ii) EXAMPLE
(iii) TEST

Each of the sub-directories is organized in "levels". To retrieve something at a higher level it is necessary to satisfy all the keys as the lower levels which lead to the required key. The three sub-directory headers are located at level zero.

The three sub-directories are stored in the same disk file. At runtime any one of these sub-directories may reside in the system's internal memory. The size of the internal directory is determined by a constant declared in the program. If a sub-directory is too large
then only a portion is read in.

For each entry in the directory, the following information is stored:

(i) The serial number in the directory.

(ii) The level at which the entry is to be placed,

(iii) The entry itself and

(iv) The record number(s) to be linked to this entry.

The serial number is used only by the editor program and has no other significance.

The level number is important because the student query is matched level by level. If any of the level does not match then the search for the query is considered unsuccessful. Details of how the directory search operates are given in section 4.4.3.

The structure of the directory record is shown in figure 4.5 and the physical organization of the DIRECTORY file is demonstrated in a sample in figure 4.6.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial No.</td>
<td>Level</td>
<td>Entry</td>
<td>Record No.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>

Figure 4.5. Structure of the DIRECTORY record.
SUBJECT DIRECTORY:
1 0 LEARN:0
2 1 GATES:0
3 2 INTRODUCTION:10
4 2 AND:0
5 3 DIAGRAM:20
6 3 DESCRIPTION:30
7 2 OR:0
8 3 DIAGRAM:60
9 3 DESCRIPTION:70
10 1 BOOLEAN ALGEBRA:0
11 2 DEFINITIONS:0
12 3 FUNCTION:300
13 0 EXAMPLE:0
14 1 K MAP:789
15 1 LOGIC DESIGN:22 43 56
16 0 TEST:0
17 1 GATES:67 89 90
18 1 K MAP:12 45 78

Figure 4.6. A Sample DIRECTORY file.
A record pointer of zero indicates that a sub-directory follows (starting from the very next line). A non-zero record pointer signifies that there is no further sub-directory and also points to the actual record(s) contained in the TEXTFILE. The level numbers link the entries which are at the same level.

The logical organization of the directory is shown in figure 4.11. The directory search techniques are discussed in section 4.4.3

4.3.3 THE RECORD-FILE:

The record-file is created and updated by the system itself. When a user runs the program for the first time, he is entered into the record-file. Each time a user runs the program the time, date, and the actual queries made are entered into the RECORD-FILE. If the student took a test, then the score for the test is also recorded. The information in the record-file is available to the teacher, who may wish to use it to analyse the students' progress and also the effectiveness of the instructional material.
The teacher also has the option of deleting all the records after he views them. When this is done all statistics and records are erased from the record-file, except the student users' names and i.d.'s.

Each record contains the following information:

(i) The student i.d. number (project-programmer number on the PDP-10).

(ii) The student's first and last names.

(iii) The date and time a query is issued.

(iv) The query itself, and

(v) the scores on all the lessons.

<table>
<thead>
<tr>
<th>Student i.d.</th>
<th>Student name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Score</td>
</tr>
<tr>
<td>Date</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Score</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Date</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Score</td>
</tr>
</tbody>
</table>

Figure 4.7. Structure of a RECORD-FILE record.
4.4 The teaching system

The teaching system is composed of four main modules (figure 4.1):

a) The INITIALIZATION module
b) The QUERY module
c) The DIRECTORY module
d) The TEACH module

The logical operation of these three modules is discussed in the following sections, the details of implementation are included in chapter 5 and the system documentation.

4.4.1 The INITIALIZATION module.

The initialization module initializes various system variables and system tables.
Figure 4.8. Flow chart of the LII/NII system.
Figure 4.9. Flow chart of the Query module.
Figure 4.10. Flow chart of the Directory module.
Figure 4.11. Logical organization of the directory.
4.4.2 The QUERY module.

The query module accepts the student user's request and performs a syntax check on it. Once the syntax has been checked, the query is separated into a list of subject-keys. The logic flow of the query module is illustrated in figure 4.9. The module creates three different lists, one for each of:

(i) LEARN  
(ii) EXAMPLE  
(iii) TEST

Depending on the query, one or more of these lists may be empty.

The syntax of the query is:

```
LEARN  subject-1, subject-2, ......., subject-9  
EXAMPLE subject-10, subject-11, ......., subject-18  
TEST   subject-19, subject-20, ......., subject-27
```

The query may include one, two or all three query keywords. The query must be typed on a single line and is terminated by a carriage return. An example of a query which requests for information on FULL BINARY ADDERS is:

```
LEARN ADDERS, BINARY, FULL
```
It is necessary to specify the subjects according to their hierarchy. The subjects must be delimited by commas; this restriction allows the system to accept subject entries which are composed of multiple words, for example:

LEARN QUINE MCCLAUSKY, RULES

As the syntax of the query language is directly related to the structure of the directory and the directory search techniques, this discussion will continue in the section on the directory.

4.4.3 The DIRECTORY module.

As mentioned in the section on the DIRECTORY file, the directory on disk is organized as three sub-directories, each one of which is further organized in LEVELS.

At one time, only one sub-directory, for any one of the following may reside in the system's internal memory:

a) LEARN
b) EXAMPLE
c) TEST

The directory module checks if the current query is from
the directory which is already in the memory. If it is, then a search is started immediately. If the query is from another sub-directory, then the directory module reads in that sub-directory and then proceeds to search it. If the sub-directory is too large to be read in at one time, then a part of it is read in and searched. If a match is not found, then another part is read in and searched, until a match is found or the sub-directory is exhausted. The directory module's keys on the "0" level to check if the complete sub-directory has been read in or not. The flow chart in figure 4.10 illustrates the logic flow of this module.

4.4.3.1 Organization of the internal directory.

If any of the sub-directories is too large to reside in the system's internal memory, it will slow down the search process, however, after a thorough study on the possible size of a sub-directory, it was decided that the possibility of such cases would be too small to justify the design of more sophisticated techniques. The present design of the internal memory will work most efficiently with sub-directories of up to 999 entries, which should be sufficient for almost any application. For larger applications it would be more advisable to break the text.
into two or more parts and thus create separate directories.

The internal directory is organized as a binary tree, each node has four fields: (1) a "same-level" link, (2) a "sub-directory" link, (3) a "record-link" and (4) the entry, as shown in figure 4.12. The nodes on the same level are linked through the "same-level" links. The "sub-directory" links point to the sub-directory for that particular entry. If the sub-directory link is NIL, that signifies that there is no further sub-directory.

The tree nodes at the last level (the leaves of the tree) are linked through the "record-links" to a list of indices which point to the records which must be retrieved from the TEXTFILE to satisfy the user request. The record-links of all other nodes are NIL.

The logical structure of the internal directory corresponding to the DIRECTORY file shown in figure 4.6 is shown in figure 4.11. Figure 4.12 shows the structure of the nodes used to store the subject entry and the record numbers in the internal memory.
<table>
<thead>
<tr>
<th>Same level</th>
<th>Entry</th>
<th>Sub-directory</th>
<th>Record</th>
<th>Link</th>
<th>Link</th>
<th>Link</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Back-link</th>
<th>Record No.</th>
<th>Forward-link</th>
</tr>
</thead>
</table>

Figure 4.12. Structure of the subject and record-pointer directory nodes.

The subject lists created by the QUERY module are passed to the DIRECTORY module. This module then searches the internal directory for the keys specified in this list, level by level. If a key is not found in the correct level, it is considered as an unsuccessful match. However, if all the keys are found in the correct levels, then the directory module creates a list of record pointers consisting of the records which must be retrieved from the TEXTFILE. Three different lists are generated, one each for:

(i) LEARN,
(ii) EXAMPLE
(iii) TEST

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
If the search is successful, but there still exists a sub-directory then this sub-directory is displayed to the student and he may then modify the query to select any one or all of the topics. In case the search was unsuccessful, then an exhaustive search is made of the entire directory to look for possible matches. All possible matches (which include any of the keys specified in the query) are displayed to the student. The student may then modify the original query.

The logic behind using the structure described above for the directory instead of an inverted list or multi list is that it is necessary for the system to be able to reference sub-directories for the subjects at each level. This can be explained easily with the help of an example. Consider the student query (refer to figure 4.11):

LEARN GATES, AND, DIAGRAM

This could be satisfied even if an inverted list, or any other structure were used by retrieving the record(s) common to all three keys. However, if the student were to ask for information on "GATES" only:

LEARN GATES

then it is necessary for the system to display the list of topics available under "GATES" so that the student may select the appropriate one. This can be done only if
links are available to the appropriate keys which constitute the sub-directory for "GATES". Moreover, if the student were to request a topic which is at a higher level, for example:

   LEARN DIAGRAM

then the system should be able to move back on the links and display the list of subjects under which "DIAGRAM" is available. This is possible with the structure described here.
4.4.4 The TEACH module.

This is the module that actually performs the task of teaching the student. It retrieves the necessary records from the TEXTFILE, accepts responses from the student and matches them with the possible responses programmed by the teacher.

The TEACH module accepts as input the lists of indices created by the DIRECTORY module. The records corresponding to these indices are retrieved one at a time from the TEXTFILE into an internal buffer. If the record is too large to be read into the buffer in one time, then only a part of it is read in, processed and then the next part read in, until the complete record has been processed.

Once the record (or a part of the record) has been read into the internal memory, it it processed according to the instructions contained in the record itself.

This module may be considered as an "Interpreter" which interprets the instructions in the TEXTFILE one at a time.
The set of instructions which may be used in the TEXTFILE is shown below in table 4.1:

**TABLE 4.1**

Set of TEXTFILE Instructions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 : n</td>
<td>(the record number)</td>
</tr>
<tr>
<td>2 :TEXT</td>
<td>(start of text)</td>
</tr>
<tr>
<td>3 :RIGHT</td>
<td>(possible correct response)</td>
</tr>
<tr>
<td>4 :WRONG</td>
<td>(possible wrong response)</td>
</tr>
<tr>
<td>5 :MATCH</td>
<td>(possible response)</td>
</tr>
<tr>
<td>6 :REPLY</td>
<td>(display text if response matches)</td>
</tr>
<tr>
<td>7 :SELECT n</td>
<td>(select and display 1 of n TEXTs)</td>
</tr>
<tr>
<td>8 :PAUSE</td>
<td>(pause while displaying text)</td>
</tr>
<tr>
<td>9 :JUMP n</td>
<td>(retrieve and process record n immediately)</td>
</tr>
<tr>
<td>10 :INSERT n</td>
<td>(retrieve and process record n after current record)</td>
</tr>
<tr>
<td>11 :CALL n</td>
<td>(call an external routine)</td>
</tr>
<tr>
<td>12 :RUN name</td>
<td>(Run a systems program)</td>
</tr>
<tr>
<td>13 /END</td>
<td>(end of record)</td>
</tr>
</tbody>
</table>
The TEACH module uses the following algorithm:

BEGIN TEACH
WHILE more records to process DO
  WHILE more of present record DO
    READ in the record into the internal buffer.
    WHILE not end of buffer or end of record DO
      decode next instruction from buffer.
      CASE instruction of:
        :TEXT - display text to student.
        :RIGHT,
        :WRONG,
        :MATCH - READ and match students response.
        :SELECT n - randomly select 1 of n :TEXTs
        :PAUSE - wait till student types
          <RETURN>.
        :JUMP n - immediately retrieve and
          process record n.
        :INSERT n - retrieve and process record n
          after current record.
        :CALL name - call an external subroutine.
        :RUN name - run a systems program.
        /END - end of record.
    END CASE
  END WHILE
END WHILE
END WHILE
END TEACH

The block diagram corresponding to this algorithm is shown in figure 4.13. A detailed description of the CASE performed on the instruction is given in the following pages.
Figure 4.13. Block diagram of the TEACH module.
4.4.4.1 The :TEXT instruction.

All text following this instruction is literally displayed on the student's terminal.

4.4.4.2 The :RIGHT instruction.

When this mark is encountered in the internal buffer it causes the system to wait for the student to type in a response. The system does not issue any prompt to the student. It is assumed that the teacher included the necessary prompt in the preceding :TEXT instruction. The response typed in by the student is matched with the literal following the :RIGHT instruction in the buffer. If a reasonable match is found, then all successive :RIGHT, :WRONG and :MATCH instructions are skipped and in the absence of a :REPLY instruction the system displays a randomly selected message telling the student that the response was acceptable.

However, if the student's response did not match with what follows the :RIGHT instruction, then the next instruction is checked. If this is also a :RIGHT, :WRONG or :MATCH instruction, then the student's response is matched with this. This process is repeated until either a match is found or there are no more possible responses.
to match with. If the latter happens then the system displays a randomly selected message saying that the student's response was not correct, and the student is asked to try again. The student may not proceed until a match has been made. Once a match has been found, all succeeding :RIGHT, :WRONG and :MATCH instructions are skipped until another :TEXT, :SELECT or /END instruction is found.

4.4.4.3 The :WRONG instruction.

This operates exactly like the :RIGHT instruction, the only difference being that if a match is obtained and the teacher did not follow this instruction with a :REPLY instruction, then the system displays a randomly selected message saying that the student was wrong. Then the student is asked to try again. The student may not proceed until a match has been made.

4.4.4.4 The :MATCH instruction:

This instruction is also similar to the :RIGHT and :WRONG instructions except that the system does not print out any message if the student's response matches with the teacher's. However, if a :REPLY instruction follows,
it is displayed to the student.

4.4.4.5 The :REPLY instruction.

This instruction may only be used immediately after a :RIGHT, :WRONG or :MATCH instruction. The literal following the :REPLY instruction is displayed to the student only if the previous instruction caused a successful match. Otherwise it is ignored.

4.4.4.6 The :JUMP n instruction.

The :JUMP instruction causes the system to branch off and retrieve a specified record (or a list of records). The :JUMP instruction may be followed by a list of one or more record pointers. As soon as this instruction is encountered, the system stops processing the current record, inserts the new list of records into the list which was created by the DIRECTORY module, and then goes and retrieves the first of these records. If the last pointer in the :JUMP instruction is a zero then it indicates that after the records in the :JUMP instruction have been processed, the lesson should be terminated. If this is the case, then after the records in the :JUMP instruction have been processed, control
will be passed back to the system. However, if the last pointer in the :JUMP instruction is not zero, then after processing the records indicated in the :JUMP instruction the system continues to follow the original sequence.

If the :JUMP instruction is used after a :TEXT instruction, then it will cause an unconditional branch to whatever sequence of records is specified.

If the :JUMP instruction follows a :RIGHT, :WRONG or :MATCH instruction (with possibly a :REPLY instruction in between), then it will cause a conditional branch, the sequence being altered only if the previous instruction resulted in a match. This lets the teacher control the logical sequence of the lesson, depending on the responses obtained from the student.

4.4.4.7 The :INSERT n instruction.

This instruction is similar to the :JUMP instruction, except that it does not cause an immediate branch. Instead, the specified record pointers are entered into the record retrieval list immediately following the current record. The current record is completely processed before the sequence is changed. If the :INSERT list terminates with a zero, this will cause
the lesson to terminate after the list of the inserted records has been processed.

4.4.4.8 The :PAUSE instruction.

This instruction may be inserted in the lesson at various places. When encountered this causes the system to print out the following message:

PRESS RETURN TO CONTINUE . . .

and wait until the student types a carriage return. This also provides the student with an exit point. If the student wishes to skip the current topic, he may type the letter S (to Skip) which causes the current record to be skipped. The student may also type the letter Q (to Quit) which will terminate the lesson.

4.4.4.9 The :CALL instruction.

This instruction allows the teacher to call external subroutines written by him. After the subroutine has been executed, control is returned back to the instruction after the :CALL instruction. The external subroutines must be written in the PASCAL. Details on how to load the external subroutine with the system are given in the teacher's manual.
4.4.4.10 The :RUN instruction.

This is similar to the :CALL instruction. Only, instead of calling a subroutine written by the teacher, it calls on a systems program. After control has been passed to the systems program it cannot be returned back to the textfile, or to the teaching system. In view of this, the :RUN instruction should only be used at the termination of the lesson, if it is to be used.

4.4.4.11 The /END instruction.

This signifies the end of the current record. When this is encountered, the system moves on and retrieves the next record from the TEXTFILE, following the sequence of the index list prepared by the directory module, and possibly modified by the :JUMP and/or :INSERT instructions.

The process of retrieving records terminates when a zero index is encountered in the index list. This fact may be used in the :JUMP and :INSERT instructions to terminate a lesson if necessary.
4.5 The Editor

As the design and implementation of the text editor is not directly related to the topic of this thesis, it has not been included here. However, details on the use of the editor are included in appendix A.

The editor allows the teacher to add new text to the TEXTFILE, edit the existing text or delete text. It also includes facilities to create, edit and maintain the system directory.
CHAPTER V

FUTURE EXPANSION AND MAINTENANCE

The design and implementation of the LEARN CAI system described in chapter 4 is complete as presented. However, there are various features which can be added to the system, and there are some existing features which can be extended to make the system more effective. Some possible expansions are discussed below.

5.1 Possible expansions

5.1.1 Use of Artificial Intelligence

As mentioned in chapter 3, the question-answer dialog with the student is limited due to the fact that the computer cannot process complex responses without the aid of complex natural-language processors. One of the improvements that could be made to the existing system is to provide such facilities through the use of artificial intelligence techniques. This could be done by writing routines which are capable of this or by interfacing the system with an existing natural-language processor.
system. Such an extension would provide for more natural dialogs between the CAI system and the student.

5.1.2 Interface between the TEXTFILE and user-written routines

At present the parameters that can be passed between the TEXTFILE and the user-written routines are limited to two arrays. It would greatly extend the system's capabilities if this restriction were to be removed.

5.1.3 The Query and Directory-search techniques

The directory search techniques used are most appropriate to this application. However, many searches fail simply because of the wrong ordering of the subjects in a query. It would greatly increase the searching capabilities of the system if the query were made independent of the subject ordering. For example, if a user wants to retrieve information on FULL BINARY ADDERS he should be able to do so with any one of the following queries:

LEARN FULL BINARY ADDERS

LEARN BINARY ADDERS, FULL
This would of course require complex artificial intelligence techniques to implement.

5.1.4 Report generation

At present, the system computes the overall score on all questions asked, and keeps a record of this. This record keeping may be extended to keep more detailed records which may be used to analyse the effectiveness of the instructional material. This might include a record of all responses to a certain question so that the teacher may evaluate the usefulness of that question.

5.1.5 Extending the TEXTFILE instruction set

The TEXTFILE instruction set may be extended to include more powerful instructions, thus providing the teacher with features to better analyse the student's responses.
5.1.6 The editor

The editing facilities can be extended to provide search-and-substitute features. Also, it would be helpful to be able to copy text from an existing file.
5.1.7 Graphic capabilities

Perhaps the most important extension to the system would be the addition of graphic capabilities. This would, of course, require the use of a graphic terminal. Such capabilities could be built into the system by writing routines to do this, or it might be possible to interface the system with an existing graphics package.

5.2 Limitations of the system

From the point of view of designing a lesson, there are two limitations, or considerations which may affect the design. The first is the number of lines which can be displayed on a video terminal. The size of a "frame" is limited to about 22 lines. It is therefore important to design the instructional material so that related information is displayed in a single frame. This imposes restrictions on the design of the material.

The second limitation is concerned with the displaying of diagrams or figures. As the system does not have any graphic capabilities, all diagrams have to be made using dots or other symbols. This restricts the information which can be displayed in figures.
A third limitation, which is dependent on the computer installation is the amount of disk space available to the CAI system. The amount of disk space required is directly proportional to the amount of information that must be stored. Therefore while designing a CAI system, it will be necessary to consider the amount of disk space available to the system.

5.3 Maintainance of the system

Complete system documentation is included with the source program which should facilitate maintainance and expansion of the system.

The teacher using the system for teaching needs to be concerned with the maintainance of the different files used by the system.

The RECORD-FILE which is used to keep the students' records is created by the system. Each time a student uses the system, the use is logged into the file. After some time the file might grow quite large. In such a case the teacher has two options:

(i) Use the system editor to erase all records from the file (except the student i.d.s and the students' names).
(ii) Delete the RECORD-FILE and thus re-initialize the system. This would be necessary if a new group of students will be using the system. This file may be deleted by the monitor command:

```
DELETE RECORD.CAI
```

The DIRECTORY is created and maintained by the teacher. The system editor may be used to do this. In its original form the directory is stored on disk as DIRECT.ORY and is necessary for the teacher while making changes to the directory. The editor command DIR/LIST may be used to create a more readable copy of the directory which can be made available to the students. This file is stored on disk as DIRECT.CAI.

The TEXTFILE is created and maintained by the teacher. The system editor may be used to do this. Records may be added, edited or deleted by the editor. Details on the use of the editor are included in the teacher's manual.

The system also maintains a HELP-FILE on disk as HELP.W.I. This includes detailed explanations of all student-user commands. The teacher may add or delete information from this file by using an external text
editor.

In its present form the object program requires about 300 disk blocks for storage. The size of the textfile depends on the amount of text included in it.
INTRODUCTION

This is a computer assisted instruction system which allows an instructor to set up a data base of information which may be used to teach and/or test students on a particular subject.

To utilize the system, the instructor need not have any knowledge of programming languages or any related field.

This system provides the instructor with a set of easy to use instructions which may be used to set up the data base, called the TEXTFILE, which contains the instructional text.

The system includes a text editor of its own which makes it easier for the user who is not familiar with any of the editors provided by the computer installation. However, in some cases, users who are familiar with other editors may wish to use them for their powerful search-and-substitute features.

Before a user can type in the text into the TEXTFILE, the material must be organized so as to be broken down into logical units.

The information entered into the TEXTFILE by the instructor-user is available to the student-user through the following three system commands:

(i) LEARN subject
(ii) EXAMPLE subject
(iii) TEST subject

The LEARN command allows the student-user to access information on the subject specified in the command, if such information is available in the system's TEXTFILE and an entry for it exists in the system directory.

Similarly the EXAMPLE command may be used to look up examples about the specified subject, if any are available.
The TEST command allows the student-user to test his knowledge on the specified subject.

For more details on how to use these instructions please refer to the Student's Manual.
AN OVERVIEW OF THE SYSTEM

This CAI system allows the instructor-user to display text (which may be instructional text or questions), receive the student's responses and compare them with the possible responses programmed by the instructor. Depending on the student's response and the way the instructional material is organized, it is possible to branch off to another part of the text, or to continue in the same logical sequence.

After a "lesson" has been written (or even as it is being written), it must be broken up into logical units (called "records").

It is necessary to do this to allow branching from one part of the lesson to another part. All entry points (a point to which a branch is made) must be declared as the beginning of a new "record".

It will be advisable to look at a sample TEXTFILE. All the instructions used are self explanatory:

EXAMPLE:

1
:INSERT 3
:text IN THIS SESSION WE WILL TRY TO FIND OUT HOW MUCH YOU KNOW ABOUT BINARY NUMBERS. HOW MANY DIGITS ARE USED IN THE BINARY SYSTEM ?
:right 2
:jump 2
:right 0,1
/end
:jump 3
:2
:text WHAT ARE THE TWO DIGITS ?
:right 0,1
/end
:3
:text SUPPOSE YOU HAVE A FIVE DIGIT BINARY NUMBERS. HOW HIGH CAN YOU COUNT WITH FIVE BITS ?
:right 31
:reply YOU PASS THE TEST !
:wrong *
The logical design of this TEXTFILE is shown in figure A.1. Once the logical design of the text is complete it is an easy matter to link the different records together.

The entry point of the lesson (record 1 in this case) must be entered into the system directory (through the EDITOR program).
Q: How many digits in the binary system?

Q: What are the two digits?

Answer = 0,1

Try again

Q: How high can 5 bits count?

Answer = 0,1

other

Discussion on Binary numbers

YOU PASS !!!

Figure A.1. Logical design of the TEXTFILE.
THE SYSTEM DIRECTORY

The system directory is organized as three sub-directories, one for each of the system KEYWORDS:

a) LEARN
b) EXAMPLES
c) TEST

All text contained in the TEXTFILE must be accessible through the system directory. The system EDITOR program may be used to make entries in the directory.

Each of the sub-directories is organized into "levels". All main subjects are located at level 1. If any subject has sub-topics, then these sub-topics are entered in the next level (level 2). If any of the subjects at level 2 have sub-topics then these may be entered in level 3 and so on until a maximum of 10 levels.

The first logical record for each entry must be specified in the directory. If a subject has a sub-directory of its own then it must be given a record number of zero. A sample directory is shown below:

<table>
<thead>
<tr>
<th>SR</th>
<th>LEVEL</th>
<th>SUBJECT</th>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>LEARN:0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>ADDERS:0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>BINARY:0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>HALF:25</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>FULL:33 34</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>DECIMAL:36</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>BCD:36</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>GATES:0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>AND:56</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>OR:57</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>INVERTER:59 60 61</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>NOT:59</td>
<td></td>
</tr>
</tbody>
</table>

The serial numbers are used only by the editor program while editing the directory. They have no other significance.
The record numbers are assigned by the user when he organizes the records in the TEXTFILE. The level must be assigned by the user when he organizes the directory. The sample directory shows that the information on "ADDERS" is included in records 25, 33, 34 and 36. If a student-user request information on "ADDERS" without further qualification, all of these records will be displayed. However, if the student requests information about "ADDERS, BINARY, FULL" then only records 33 and 34 will be displayed.

It is up to the instructor-user to set up a well organized directory.

Note that the system directory only provides an entry point into the TEXTFILE. Once control has been passed to the TEXTFILE, the instructor may use :JUMP and :INSERT instructions to move all over the TEXTFILE.
THE EDITOR

This CAI system provides its own editor program to create and maintain the TEXTFILE and the system DIRECTORY. To run the editor use the MONITOR command:

RUN EDITOR

The editor prompts with:

>>

At this level the editor's SYSTEM COMMANDS may be used.

THE EDITOR COMMANDS

The editor commands are classified into the following categories:

a) SYSTEM commands
b) TEXTFILE commands
c) EDIT MODE commands
SYSTEM COMMANDS:

1) ERASE  -  To erase statistics from the system record file.
2) DIR/LIST - To create a readable copy of the directory.
3) DIRECT - To edit the directory.
4) NEW    - To enter TEXTFILE and create a new record.
5) ALTER  - To enter TEXTFILE and edit an old record.
6) DELETE - To delete an old record.

TEXTFILE COMMANDS:

<table>
<thead>
<tr>
<th>Command</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>T</td>
</tr>
<tr>
<td>RIGHT</td>
<td>RI</td>
</tr>
<tr>
<td>WRONG</td>
<td>W</td>
</tr>
<tr>
<td>MATCH</td>
<td>M</td>
</tr>
<tr>
<td>REPLY</td>
<td>RE</td>
</tr>
<tr>
<td>SELECT</td>
<td>S</td>
</tr>
<tr>
<td>PAUSE</td>
<td>P</td>
</tr>
<tr>
<td>NOPAUSE</td>
<td>N</td>
</tr>
<tr>
<td>JUMP</td>
<td>J</td>
</tr>
<tr>
<td>INSERT</td>
<td>I</td>
</tr>
<tr>
<td>RUN</td>
<td>RU</td>
</tr>
<tr>
<td>CALL</td>
<td>C</td>
</tr>
<tr>
<td>END</td>
<td>E</td>
</tr>
<tr>
<td>QUIT</td>
<td>QUIT</td>
</tr>
</tbody>
</table>

- to enter :TEXT
- to enter :RIGHT response
- to enter :WRONG response
- to enter :MATCH response
- to enter :REPLY to match
- to enter :SELECT n
- to enter :PAUSE
- to stop pause feature
- to enter :JUMP n1 n2 n3 ...
- to enter :INSERT n1 n2 n3 ...
- to enter :RUN program name
- to enter :CALL procedure name
- to end current record
- to abort current edit
EDIT MODE COMMANDS:

The following commands may be used to insert and edit text while in TEXTFILE mode, except for END, PAUSE and SELECT.

I    - Insert characters until terminated by an ALTMODE.
nD   - Delete n characters to the right of curser. If n is negative then delete n characters to the left.
W    - Move curser over one word to the right.
L    - Move curser to beginning of line.
nL   - Move curser over n lines to the right.
-nL  - Move curser over n lines to the left.
P    - Print current line.
nP   - Print n lines to the right of curser.
-P   - Print the entire record being edited.
E$   - (letter E followed by an ALTMODE). End edit.
Q    - Abort edit. User asked to confirm with "Y".
TEXTFILE INSTRUCTIONS

Each record in the TEXTFILE must begin with a record number and end with an end-of-record mark.

A record may contain the instructions shown in the table below:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>:n</td>
<td>(the record number)</td>
</tr>
<tr>
<td>:TEXT</td>
<td>(start of text)</td>
</tr>
<tr>
<td>:RIGHT</td>
<td>(possible correct response)</td>
</tr>
<tr>
<td>:WRONG</td>
<td>(possible wrong response)</td>
</tr>
<tr>
<td>:MATCH</td>
<td>(possible response)</td>
</tr>
<tr>
<td>:REPLY</td>
<td>(display text if response matches)</td>
</tr>
<tr>
<td>:SELECT n</td>
<td>(select and display 1 of n TEXTs)</td>
</tr>
<tr>
<td>:PAUSE</td>
<td>(pause while displaying text)</td>
</tr>
<tr>
<td>NOPAUSE</td>
<td>(stop auto pause)</td>
</tr>
<tr>
<td>JUMP n</td>
<td>(retrieve and process record n immediately)</td>
</tr>
<tr>
<td>INSERT n</td>
<td>(retrieve and process record n after current record)</td>
</tr>
<tr>
<td>CALL n</td>
<td>(call an external routine)</td>
</tr>
<tr>
<td>RUN name</td>
<td>(Run a systems program)</td>
</tr>
<tr>
<td>/END</td>
<td>(end of record)</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Using an external editor:

If using an editor other than the one provided by this CAI system the following rules must be observed:

(i) The TEXTFILE instructions must begin in the first column of a new line.
(ii) The text itself must begin in the ninth column of every line.

NOTE: Throughout this document whenever a dialog is shown between the computer and the user, the response from the user is underlined.
DIRECT [SYSTEM command]

SYNTAX:

>> DIRECT

This command may be used to edit the system directory. When given this command, the editor responds with the message shown below:

>> DIRECT
Type CTRL-Z to end edit.
ENTER:
Sr., Level, Subject, Record s *

The new entry is entered at the serial number specified in the above instruction. If an entry already exists at the specified serial number then the new entry is placed immediately after that.

For details on the LEVEL and the RECORD NUMBERS refer to the section on the DIRECTORY.

When the editor is in the DIRECT mode, and prompts with an asterisk the user may delete a record by the command:
*DELETE n
or
*D_N

The serial numbers in the directory may be resequenced by the command:
*RENUMBER
or
*REN
DIR/LIST  [SYSTEM command]

SYNTAX:

>> DIR/LIST

This creates a readable listing of the system directory on the file DIRECT.CAI.

ERASE  [SYSTEM command]

SYNTAX:

>> ERASE

This instruction may be used to reinitialize the system record-file which keeps statistics on the students. Everything is erased from the record-file except the student i.d.s and names. If it is desired to delete the record-file entirely then the following MONITOR command may be used:

DELETE RECORD.SYS

NEW  [SYSTEM command]

SYNTAX:

>> NEW

Allows the user to enter a new record into the TEXTFILE. May be followed by any of the EDIT MODE commands. More details may be found in the section on TEXTFILE instructions.
ALTER n [SYSTEM command]  
SYNTAX:  
>> ALTER n  

Allows user to edit an existing record from the TEXTFILE. The editor opens the TEXTFILE at record n. The user may then use any of the EDIT MODE commands to edit the record.

DELETE N [SYSTEM command]  
SYNTAX:  
>> DELETE n  

May be used to delete a record from the TEXTFILE.

PROCEDURE [SYSTEM command]  
SYNTAX:  
>> PROCEDURE  
PROCEDURE NAME ? name  
NOW PLEASE TYPE THE FOLLOWING MONITOR COMMAND:  
.DO LOAD name  
.DO LOAD name
THE TEXTFILE COMMANDS

: n    (The record number).

SYNTAX:

To create a new record use the editor command:

>>NEW n

Where n is the new record number. If no number is specified in the command then the new record is appended to the end of the TEXTFILE. When issued this command, the editor responds with an asterisk, signifying it is ready to accept any of the TEXTFILE commands.

EXAMPLE:

> NEW

* 

Each record in the TEXTFILE must be assigned an integer number. The only restrictions on the assignment of these numbers are:

(i) There should be no duplicates. If there are any duplicates, only the first record will be accessible by the system.

(ii) The numbers must be in ascending order in the TEXTFILE. They need not however be consecutive. In fact it is advisable to use a step of 10 (or any other appropriate step) to allow insertion of records in the future without affecting the physical organization of the TEXTFILE. However, it must be understood that the physical organization of the TEXTFILE has nothing to do with its logical organization. A record at the end of the file may be logically linked to a record at the beginning of the file without any problem.

When using the system editor, the user does not need to be concerned with the above restrictions. Initially the editor assigns record numbers in steps of 10. While inserting records between two existing records, however, it is up to the user to specify the new record number.
TEXT [TEXTFILE command]

SYNTAX:

To enter text into the current record use the TEXTFILE command:

*TEXT (the single letter T is sufficient)
The editor will respond by typing:

:TEXT
followed by a carriage return.
Now any of the EDIT MODE commands may be used.

EXAMPLE:

> NEW
*T
:TEXT
  This is a new record !

This instruction may be used to enter text which is to be displayed on the student's terminal. The text will be displayed exactly as it is entered, including all blanks and blank lines. While using the system editor the text may begin in the first column and continue for as many columns as necessary (although it is advisable to keep it less than or equal to 80 columns to make it readable).

IMPORTANT: The carriage return at the end of the text is not displayed on the student's terminal. This is useful if it is desired to have the cursor wait on the same line as the text, so that the student may enter a response. However, if this is not desired, then the text must be terminated with an extra carriage return which will force the system to print out a carriage return.
:RIGHT [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:
   *RIGHT (the letters RI are sufficient)
The editor will respond with:
   :RIGHT
followed by a carriage return.
Now any of the EDIT MODE commands may be used.

EXAMPLE:

> NEW
   *T
   :TEXT
   HOW DO YOU SPELL 19 ?
   *RI
   :RIGHT
   NINETEEN
   *

This instruction may be used to enter a possible correct-response. The text for this instruction must be typed on a single line.

When this mark is encountered it causes the system to wait for the student to type in a response. The system does not issue any prompt to the student. It is assumed that the necessary prompt was included in the preceding :TEXT instruction. The response typed in by the student is matched with the literal following the :RIGHT instruction. If a reasonable match is found then all successive :RIGHT, :WRONG and :MATCH instructions are skipped and in the absence of a :REPLY instruction the system displays a randomly selected message telling the student that the response was acceptable.

However, if the students response did not match with what follows the :RIGHT instruction, then the next instruction is checked. If this is also a :RIGHT, :WRONG or :MATCH instruction then the students response is matched with this. This process is repeated until either a match is found or there are no more possible responses to match with. If the latter happens then the system displays a randomly selected message saying that the student's response was not correct, and the student is asked to try again. The system does not proceed to the next :TEXT instruction until a match has been made. Once a match has been found, all succeeding :RIGHT, :WRONG and
MATCH instructions are skipped until another TEXT, SELECT or /END instruction is found.

Using the "*":

An asterisk may be used to match with "anything" in the student's response. For example if the expected response from a student is "YES" the the following instruction will accept anything that starts with the letter "Y":

*RI
:RIGHT
Y*

The asterisk may be used anywhere in the instruction. It may also be used more than once. For example the following instruction will match any response as long as it contains the word "YES" followed by the word "IS":

*RI
:RIGHT
*YES*IS*

Thus if the student types any one of the following responses it will be considered a correct response by the system:

YES IT IS
YES IT IS CORRECT
YES THAT IS RIGHT

However, care must be taken while using this because if the student types:

YES THAT IS NOT CORRECT

The system will still consider it a correct response!

Matching parameters:

Parameters used in procedure calls may be used in a :RIGHT instruction. Refer to the section on the :CALL instruction for notes on syntax and use.
:WRONG  [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:

*WRONG  (the letter W is sufficient)

The editor will respond with:

:WRONG

followed by a carriage return.

Now any of the EDIT MODE commands may be used.

EXAMPLE:

> NEW
*T
:TEXT
HOW DO YOU SPELL 19 ?
*RI
:RIGHT
NINETEEN
*W
:WRONG
NINTEEN
*

This instruction may be used to enter a possible wrong-response. The text for this instruction must be typed on a single line.

When this mark is encountered it causes the system to wait for the student to type in a response. The system does not issue any prompt to the student. It is assumed that the necessary prompt was included in the preceding :TEXT instruction. The response typed in by the student is matched with the literal following the :WRONG instruction. If a reasonable match is found then all successive :RIGHT, :WRONG and :MATCH instructions are skipped and in the absence of a :REPLY instruction the system displays a randomly selected message telling the student that the response was unacceptable.

However, if the students response did not match with what follows the :RIGHT instruction, then the next instruction is checked. If this is also a :RIGHT, :WRONG or :MATCH instruction then the students response is matched with this. This process is repeated until either a match is found or there are no more possible responses to match with. If the latter happens then the system displays an randomly selected message saying that the
student's response was not correct, and the student is asked to try again. The system does not proceed to the next :TEXT instruction until a match with a :RIGHT or :MATCH instruction has been made. Once such a match has been found, all succeeding :RIGHT, :WRONG and :MATCH instructions are skipped until another :TEXT, :SELECT or \END instruction is found.

Using the "*":

An asterisk may be used to match with "anything". For details on this refer to the section on the :RIGHT instruction.

EXAMPLE:

```
> NEW
*T
:TEXT
HOW DO YOU SPELL 19 ?
*R
:RIGHT
NINETEEN
*W
:WRONG
*
*
```

This will cause any answer which is not NINETEEN to be considered a wrong response.

Matching parameters:

Parameters used in procedure calls may be used in a :WRONG instruction. Refer to the section on the :CALL instruction for notes on syntax and use.
MATCH [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:
   *MATCH (the letter M is sufficient)
The editor will respond with:
   :MATCH
followed by a carriage return.
Now any of the EDIT MODE commands may be used.

EXAMPLE:

   > NEW
   *T
   :TEXT
   HOW DO YOU SPELL 19 ?
   *R
   :RIGHT
   NINETEEN
   *M
   :MATCH
   NINTEN

This instruction may be used to trap a wrong answer without giving the student another chance to answer. If the match option is not used the the student will be asked to try again and again. However, if the match option is used, then the student will be allowed to proceed further even if the response was not correct.

Using the "*":

The asterisk may be used to match "anything" in the student's response. For details on this refer to the section on the :RIGHT instruction.

As mentioned in the section on the :RIGHT instruction, the system proceeds sequentially through all the :RIGHT, :WRONG and :MATCH instructions following a :TEXT instruction. The first match with any of these terminates the matching process and the rest of the instructions are skipped till the next :TEXT instruction. Therefore the last instruction in this group is executed only if all others failed to match. If it is not desired to have the student asked the question again, the :MATCH instruction may be used as the last of the set (using an asterisk) to trap "all" responses. This is illustrated in the following example:
EXAMPLE:
> NEW
*T
HOW DO YOU SPELL 19 ?
*R1
:RIGHT
NINETEEN
*L
:MATCH
*

In this case even if the student gives a wrong answer, the same question will not be asked again.

Matching parameters:

Parameters used in procedure calls may be used in a :MATCH instruction. Refer to the section on the :CALL instruction for notes on syntax and use.
:REPLY [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:

    *REPLY (the letters RE are sufficient)

The editor will respond with:

    :REPLY

followed by a carriage return.

Now any of the EDIT MODE commands may be used.

EXAMPLE

    > NEW
    *T
    :TEXT
    HOW DO YOU SPELL 19 ?
    *RI
    :RIGHT
    NINeteen
    *RE
    :REPLY
    GOOD !
    *M
    :MATCH
    NINeteen
    *RE
    :REPLY
    NO, THAT IS NOT CORRECT !

When a student's response is successfully matched with a :RIGHT or :WRONG instruction then the system displays an appropriate message informing the student whether the response was correct or not. These messages are selected at random from a list of messages stored as part of the system. However, if the student's response matches with a :MATCH instruction then no message is displayed by the system. In some cases the instructor creating the text may wish to display a message even if the :MATCH instruction resulted in a match, or to display a more meaningful message than are displayed by the system on a :RIGHT or :WRONG instruction.

This may be done by the use of the :REPLY instruction. Whenever a :RIGHT, :WRONG or :MATCH results in a match with the student's response, if the next instruction is :REPLY then the text following this is displayed to the student instead of the system's own message.
The :REPLY instruction is similar to the :TEXT instruction, except the text following the :REPLY is displayed only if the previous instruction resulted in a match.
:SELECT n [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:

*SELECT (the letter S is sufficient)
OUT OF ? n

where n is an integer between 2 and 9.

This instruction may be used before a group of :TEXT instructions to randomly select one of the following "n" :TEXTs for displaying to the student.

All instructions following a :TEXT instruction, up to the next :TEXT instruction are considered as part of the :TEXT instruction and are skipped with it when one of the "n" :TEXTs is being selected.

EXAMPLE

(This example is displayed as it appears in the TEXTFILE. While it is being created, the instructions will appear on new lines).

: 30
:TEXT HOW DO YOU SPELL
:SELECT 3
:TEXT 19
:RIGHT NINETEEN
:REPLY VERY GOOD !
:TEXT 77
:RIGHT SEVENTY*SEVENTY
:TEXT 99
:RIGHT NINETY*NINETY
:TEXT WELL, NOW WE KNOW HOW WELL YOU CAN SPELL !
-END

The above record will cause the first :TEXT to be displayed, followed by only one of the following three :TEXTs, selected at random, followed by the last :TEXT.

If the program is run once and then run again, a different combinations may be expected.

NOTE: The :SELECT will not cross over record boundaries.
:RUN name  [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:
*RUN  (the letters RU are sufficient)

    PROGRAM NAME ?  name

the program name may be typed in using the EDIT MODE instructions.

This instruction may be used to call an external systems program provided by the computer installation. "name" may be the name of any such systems program. Only the first six characters are significant.

When this instruction is encountered in the TEXTFILE, a call is made to the specified program. Once control is passed to the external program, it cannot be returned to the CAI system. On exit from the external program control passes to the MONITOR.

If the specified program is not found or any other error is caused due to this call, control is passed to the MONITOR.
CALL name [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:

*CALL (the letter C is sufficient)
PROCEDURE NAME ? name

Now any of the EDIT MODE commands may be used.

This instruction allows the user to call on external subroutines written by the user in the PASCAL language.

Only the following two parameters may be passed between the TEXTFILE and the PASCAL procedure:

(i) I : ARRAY [0..9] OF INTEGER;
(ii) C : ARRAY [0..9] OF PACKED ARRAY [1..30] OF CHAR;

Assignments may be made to the parameters in the CALL instruction as shown below. The syntax of the call is:

CALL name (ZERO, parameter=parameter/literal,
parameter=parameter/literal,...)

where

name may be any PASCAL procedure name.
(only first nine characters are significant).
the following 20 parameters may be used:

I[0]
I[1]
.
.
I[9]
C[0]
C[1]
.
.
C[9]

For the I parameters, the literal may be an integer number upto nine digits. For the C parameters, the literal may be upto 30 characters long, enclosed in single quotation marks.

The ZERO option:
The ZERO specification, if included in the call causes the array I to be initialized to zeroes, and array C to be initialized to blanks before the call is made.

If the ZERO specification is not used in the call then any previous values (from a previous call to ANY procedure) will be retained. This allows values to be retained between procedure calls, and also between calls to different procedures.

Parameters which are assigned values in the call will enter the procedure with these new values.

Parameters which are not assigned any values (i.e. do not appear in the call) will retain previous values, unless the ZERO specification is included.

EXAMPLE:

If the following call is included in the TEXTFILE, it will first zero out array I, blank out array C, and then assign the integer 3 to I[0], the literal ABC to C[0] and so on.

:CALL KMAP (ZERO, I[0]=3, I[3]=8, C[0]="ABC")

The following call without "ZERO" will retain the values of I[0] and I[3] assigned above

:CALL KMAP (C[0]="XYZ")

A call without any parameters allows all parameters to retain their values:

:CALL KMAP

Using parameters in :RIGHT, :WRONG and :MATCH instructions:

The parameters passed back from a procedure may be used in the match instructions. The parameters when included in the match instructions must be preceded by an equal sign, and no blanks must be used in between.

EXAMPLE:

:RIGHT =I[4]

This will cause a match if the student's response is the same as the literal of the integer contained in the parameter I[4].

Writing the procedures:
An external editor must be used to create the files containing user written procedures.

Any program name may be used, and any procedure name may be used. The following TYPE declarations must be included in the external procedure. (type and variables names may be substituted with more meaningful names).

(*$E+*)

PROGRAM X, Y;

TYPE
  I_ARRAY : ARRAY [0..9] OF INTEGER;
  C_ARRAY : ARRAY [0..9] OF PACKED ARRAY [1..30] OF CHAR;
  SCORE_TYPE : RECORD
    QUESTIONS : INTEGER;
    RIGHT : INTEGER;
    WRONG : INTEGER;
  END;

PROCEDURE Y (VAR I : I_ARRAY; VAR C : C_ARRAY;
  VAR SCORE : SCORE_TYPE);

BEGIN
  (* the procedure body *)
END;

BEGIN
END.

SYNTAX:

Once the procedure has been written and complied without errors, it may be added to the CAI systems library by the following editor (SYSTEM MODE) command:

```
>>PROCEDURE
```

The editor will respond with:

```
PROCEDURE NAME ?
```

after the user types in the procedure name the system will type:

```
NOW PLEASE TYPE THE FOLLOWING MONITOR COMMAND:
```

```
.DO LOAD name
```

where name is the name of the new procedure. This will load the new procedure along with the system.
::PAUSE   [TEXTFILE command]

SYNTAX:

*PAUSE

This instruction may be inserted in the lesson at various places. When encountered this causes the system to print out the following message:

> PRESS RETURN TO CONTINUE . . .

and wait until the student types a carriage return. This also provides the student with an exit point. If the student wishes to skip the current topic he may type the letter S (to Skip) which causes the current record to be skipped. The student may also type the letter Q (to Quit) which will terminate the lesson.

If the :PAUSE is not used then the system itself inserts a pause after 22 lines have been displayed.

::NOPAUSE   [TEXTFILE command]

SYNTAX:

*NOPAUSE

This instruction if used, disables the pause which is inserted automatically by the system after 22 lines. The auto pause may be enabled by the use of a :PAUSE instruction.
**SYNTAX:**

Use the following TEXTFILE command:

\[
*\text{JUMP} \quad \text{(the letter J is sufficient)}
\]

:JUMP n1 n2 n3 ....  

any of the EDIT MODE commands may be used to enter n1, n2, n3 ... which are record numbers.

This instruction may be used to logically link records together. The :JUMP instruction may be followed by a list of one or more record numbers. As soon as this instruction is encountered, the system stops processing the current record, and jumps to the beginning of the first record specified in the instruction.

If the last record number in the :JUMP instruction is a zero then it indicates that after the records in the :JUMP instruction have been processed, the lesson should be terminated. If this is the case then after the records in the :JUMP instruction have been processed, control will be passed back to the system. However, if the last record number in the :JUMP instruction is not zero then after processing the records indicated in the :JUMP instruction the system continues to follow the original sequence as specified in the directory.

If the :JUMP instruction is used after a :TEXT instruction then it will cause an unconditional branch to whatever sequence of records is specified.

If the :JUMP instruction follows a :RIGHT, :WRONG or :MATCH instruction (with possibly a :REPLY instruction in between) then it will cause a conditional branch, the sequence being altered only if the previous instruction resulted in a match. This lets the instructor control the logical sequence of the lesson, depending on the responses obtained from the student.

**EXAMPLE:**

A record in the TEXTFILE using the :JUMP instruction may look like this:

\[
= 35
\]

=TEXT SELECT BETWEEN THE FOLLOWING OPTIONS:

STOP
CONTINUE
REPEAT
MATCH S*
JUMP 0
MATCH C*
REPLY OKAY! HERE WE GO!
MATCH R*
JUMP 25 27 0
/END
:INSERT n1 n2 n3 .... [TEXTFILE command]

SYNTAX:

Use the following TEXTFILE command:

*INSERT (the letter I is sufficient)
:INSERT n1 n2 n3 ....

any of the EDIT MODE commands may be used to enter n1, n2, n3 ... which are record numbers.

This is the same as the :JUMP instruction with the important difference that it does not cause an immediate jump. A jump to the first record in the instruction is made only after the current record has been processed. Apart from this there is no difference between the two instructions.
APPENDIX B

LEARN
COMPUTER-ASSISTED INSTRUCTIONAL SYSTEM

STUDENT'S USER MANUAL
INTRODUCTION

This is a Computer-Assisted Instruction (CAI) system called LEARN. The objective of this system is to provide an easy way of learning about a particular subject and then testing the knowledge thus acquired. The three options available to the student user are:

a) To use the system command LEARN and retrieve information about a particular topic.
b) Use the command EXAMPLES to retrieve examples pertaining to a particular topic.
c) Use the command TEST to test the user's knowledge on a particular topic.

The LEARN program may be run by the MONITOR COMMAND:

.RUN LEARN

The first time a user runs LEARN the system requests the user's name and enters it into the system's records. After this every time the program is run, it greets the student by his/her name.

The program keeps a record of all the requests typed by the user, and also all scores on questions asked. Each time the program is run, the user is asked if he/she wishes to look at his/her previous record. The user is given the option to look at the entire record, or the record pertaining to the previous session only. After this, the system prompts the user with:

Waiting for your instructions . . .

At this point the user may issue a request for information on a certain topics, or to be tested on a certain topic.

NOTE: In this manual the following convention is used to differentiate between the message typed by the computer and response from the user: The response typed by a user is underlined, whereas the messages typed by the computer are not.
EXAMPLE:

The first dialog may look like this:

```
RUN_TEACH
Hi! Seems like this is the first time we are meeting!
Will you please type in your first name: TAHIR
Last name: MUFTI
If you want to find out how to talk to me, type HELP
Waiting for your instructions . . .
>> FINISH

It was nice seeing you Tahir!
```

EXAMPLE: A second run may look like this:

```
RUN_TEACH
Hi Tahir! Nice day isn't it?
Do you want to look at your past record? YES
You have no previous record.
Waiting for your instructions . . .
>> FINISH

See you later!
```
The LEARN system consists of the following three main parts:

1 - The system program which may be run by the monitor command:

    RUN TEACH

2 - The system library which includes information about certain subjects.

3 - The system directory, which is a list of all topics available in the system library. The system directory is available on disk and may be printed on the line-printer by the monitor command:

    PRINT DIRECT.CAI

or it may be typed out on a terminal by the monitor command:

    TYPE DIRECT.CAI

It is helpful, although not necessary, to have the directory available when using the system.

The LEARN system has the following command:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEARN &lt;subject&gt;</td>
<td>Retrieves information on &lt;subject&gt;</td>
</tr>
<tr>
<td>EXAMPLE &lt;subject&gt;</td>
<td>Retrieves examples about &lt;subject&gt;</td>
</tr>
<tr>
<td>TEST &lt;subject&gt;</td>
<td>Presents a quiz/test on &lt;subject&gt;</td>
</tr>
<tr>
<td>END</td>
<td>Exits from the program</td>
</tr>
<tr>
<td>FINISH</td>
<td>&quot;</td>
</tr>
<tr>
<td>QUIT</td>
<td>&quot;</td>
</tr>
<tr>
<td>HELP</td>
<td>Displays a list of commands.</td>
</tr>
<tr>
<td>MORE HELP</td>
<td>Prints out a HELP file which is also available on disk and may be obtained by the monitor command: PRINT HELP.CAI</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>Lets the user type in comments to the supervising instructor.</td>
</tr>
</tbody>
</table>
The LEARN command.

SYNTAX:

LEARN subject-1, subject-2, ......, subject-9

This command is used to request information on a certain subject. The subject must be specified in an hierarchical order, according to the corresponding entry in the system directory.

EXAMPLE:

To get information on FULL BINARY ADDERS the command would be:

LEARN ADDER, BINARY, FULL

If there is any information on this subject, it will be displayed, otherwise the system will inform the user that no information is available on this subject.

If a subject directory is not available, the LEARN command can be used to make the system display a sub-directory for a particular topic. For example:

LEARN ADDERS
Want to see what is available under "ADDERS"? YES
  BINARY
  DECIMAL
  OCTAL
  HEX
  What would you like to see?
  >> BINARY
Want to see what is available under "ADDERS,BINARY"? YES
  FULL
  HALF
  What would you like to see?
  >> FULL

If there is a further break-down of topics available, the dialog will continue. If not, then the requested information will be displayed.
While text is being displayed, the system pauses after every 22 lines, and displays this message:

PRESS RETURN TO CONTINUE . . .

At this time the user has the following options:

1 - Press the RETURN key to continue.
2 - Press S (for SKIP) to skip the current topic.
3 - Press Q (for QUIT) to exit from the program.
**The EXAMPLE command**

**SYNTAX:**

\[ \text{EXAMPLE subject-1, subject-2, ..., subject-9} \]

This command retrieves examples on the specified subject. The same rules as described for the LEARN command apply here.

---

**The TEST command**

**SYNTAX:**

\[ \text{TEST subject-1, subject-2, ..., subject-9} \]

This command presents a quiz (or test) on the specified subject. The same rules as described for the LEARN command apply here.

Before the quiz is presented, a list of rules is displayed to the student, informing him/her of the restriction on the format of the response. At the end of the test the score obtained is displayed. During the test the student may NOT SKIP any part. (unless the particular test has been set up to allow it).

---

**The EXPERT option:**

If the user does not wish to have the rules displayed, the EXPERT option may be entered at the end of the TEST command:

\[ \text{>>TEST GATES /EXPERT} \]
The **COMMENTS** command

**SYNTAX:**

```
COMMENTS
```

The system responds with:

```
[Message being recorded, when done type: CONTROL-Z]
```

The message may be as long as necessary. To exit, the user must type CONTROL-Z, which will also cause an exit from the program.

The **END, FINISH, and QUIT commands**

**SYNTAX:**

```
>>END
>>FINISH
>>QUIT
```

These three instructions are equivalent. They cause an exit from the program.
The HELP command

SYNTAX:

>>HELP

This displays a summary of the system commands.

The MORE HELP command

SYNTAX:

>>MORE HELP

This displays a help file giving detailed instruction on the use of the system and explaining the system commands. The help file is also available on disk. A line-printer copy may be obtained by the monitor command:

PRINT HELP.CAI

or a copy may be obtained on a terminal by the monitor command:

TYPE HELP.CAI
The system records

Each time a user uses the system, a record is kept of the date, time and the actual request typed. If any questions are asked by the system, a score is kept of the correct responses.

When the user runs the program, he/she has the option of looking at his/her previous record. The user may look at the entire record or only at the record pertaining to the last session.

The supervising instructor may erase the system record when they grow too large, therefore it is possible for the user to get a message saying he/she has no previous record.
APPENDIX C

APPLICATION OF THE LEARN CAI SYSTEM TO COMPUTER STRUCTURES AND HARDWARE
APPLICATION TO COMPUTER STRUCTURES

As part of this thesis, the LEARN CAI system has been applied towards the teaching of a course in COMPUTER STRUCTURES.

A brief description of the material included in this application and the logical organization of the material is discussed here.

The topics included in the application are listed in appendix D, a sample of the actual text is included in appendix E and sample runs are included in appendix F.

The Instructional Material

The instructional part of the application attempts to lead a student from the very basics of the binary number system, through boolean algebra, logic gates onwards to the logical design of a complete computer system.

The system directory is organized in such a manner so as to lead a new-commer to the subject step by step through the material. If the sequence is followed as presented, the student should have no problem progressing through the material.

However, if the student already has previous knowledge of some topics he may skip these topics without any concern.

For someone who already has a good knowledge of the subject, this may serve as a "reference" system. It has been attempted to keep all material sufficiently independent so as to be useful in this respect.
ORGANIZATION OF THE MATERIAL

Although a complete list of topics included in the text is given in appendix D, the material may logically be separated into the following sections:

- Number systems.
- Boolean algebra.
- Logic gates.
- Boolean functions/expressions and Minimization.
- Basic logic blocks (adders, flip flops, etc).
- Combinational circuits.
- Sequential circuits.
- Computer Structures.

The text is organized, more or less in the sequence shown above. However there may be some sub-topics which are common to more than one topic.

Number Systems:

It is expected that a new student, using the system for the first time will have some basic mathematical skills. Knowledge of the Binary number system and boolean algebra would certainly be helpful, but the system does not depend on this. The student is lead through a discussion on number systems, the most important topics being: binary, octal and hexadecimal number systems, binary subtraction by complementation and conversion between number systems.

Once the student has a fair understanding of the binary number system he should be able to proceed to the section on Boolean algebra.

Boolean Algebra:

This section attempts to define Boolean Algebra and the basic boolean functions. The basic rules of boolean algebra are presented, especially Demorgan's Theorems, and their application towards the minimization of boolean expressions is demonstrated.
Logic Gates:

This section tries to set up a parallel between the basic boolean functions and the electronic gates used to realize these functions. The student is made familiar with the graphic symbols used to designate the different gates. The functional description of the gates is presented with the help of truth tables. The information in this section is important because the following material assumes a fair knowledge of logic gates and there are frequent examples showing logic circuits which would be used to realize a particular boolean function.

Boolean Expressions/Functions and Minimization:

This section includes a discussion on how to minimize boolean expressions. First the student is shown how to minimize expressions by using the basic rules of boolean algebra. After the student appreciates the importance of recognizing familiar patterns and realizes that there are no actual hard and fast rules to be followed in this method he is introduced to the Karnaugh Map and the Quine-McCluskey methods. Examples of both methods are shown and the student is asked to join in after he has a fair knowledge of the subject.

Basic Logic Blocks (adders, flip flops etc):

This section includes a discussion on half and full adders, subtractors and the various types of flip flops. The student is familiarized with the graphic symbols, the operation and the truth tables for these basic logic blocks.

The use of these blocks is demonstrated in the text related with combinational and sequential circuits.

Combinational Circuits

Once the student has acquired basic knowledge of logic gates and adder circuits he is introduced to the designing and analysis of combinational circuits. Detailed analysis of combinational circuits such as decoders and parallel adders is given along with some design examples.
Sequential Circuits:

This is where the student realizes the importance of the flip flop. Detailed analysis of basic sequential circuits, such as serial and parallel registers, serial adders and master-slave flip flops are shown. The student is lead through a number of steps leading to the design of sequential circuits.

Computer Structures:

This section is perhaps the most important one, because all the other material was used to build up the student's knowledge to this level. After a simple introduction to the logical organization of a computer system, the student is lead through detailed discussion of the various units which comprise the computer system.

The CPU is discussed in considerable detail, including the structure of the CPU, the different control signals and the various registers included in the CPU.

A hypothetical micro-computer is set up and used to teach the basics of micro-programming.

The various means of I/O and the types of storage memories used are also discussed in fair detail. The student's previously acquired knowledge of decoders and registers is utilized while explaining the means of accessing memory locations in a primary memory.

Examples

The section on examples is organized parallel to the instructional material. Examples are included for all topics. In most cases these examples are presented while the student is progressing through the instructional material. At other times the student is told that examples are available and if he wishes, he may look at them. The examples on a particular topic are arranged in increasing order of difficulty. If there are a number of examples which fall into the same level of difficulty then one (or more) of these are selected at random, thus providing a variety. However, the student may request a particular example in which case the random selection will not be made.
Testing

Short quizzes on all the topics included in the instructional material are available. These quizzes may be used by the student to improve skills on a particular topic.

In addition to the quizzes, the student may request a "comprehensive" exam. The comprehensive exam selects random problems from randomly selected topics and thus tests the student's overall knowledge of the subject. Due to the random selection of problems, a different exam is presented each time the student attempts it. There are no restrictions imposed by the system on the number of times a student takes an exam, however, a record is kept of the scores on all exams.
APPENDIX D

COMPUTER STRUCTURES AND HARDWARE
SUBJECT DIRECTORY
SUBJECT DIRECTORY FOR COMPUTER ASSISTED INSTRUCTIONAL SYSTEM FOR COMPUTER STRUCTURES AND HARDWARE.

LEARN
GATES
  INTRODUCTION
  AND
    DIAGRAM
    DESCRIPTION
    EQUATION
    TRUTH TABLE
  OR
    DIAGRAM
    DESCRIPTION
    EQUATION
    TRUTH TABLE
INVERTER
  DIAGRAM
  DESCRIPTION
  EQUATION
  TRUTH TABLE
NAND
  DIAGRAM
  DESCRIPTION
  EQUATION
  TRUTH TABLE
NOR
  DIAGRAM
  DESCRIPTION
  EQUATION
  TRUTH TABLE
EXCLUSIVE OR
  DIAGRAM
  DESCRIPTION
  EQUATION
  TRUTH TABLE
EQUIVALENCE
  DIAGRAM
  DESCRIPTION
  EQUATION
  TRUTH TABLE
BOOLEAN ALGEBRA
DEFINITIONS
FUNCTION
FUNCTIONS
  AND
  OR
  COMPLMENTATION
  NAND
NOR
EXCLUSIVE OR
COINCIDENCE
NOTE
RELATIONS
INTRODUCTION
BASIC RELATIONS
MINIMIZATION
DEMORGAN'S THEOREM
MINTERMS
DEFINITION
EXPLANATION
ADDERS
HALF
DESCRIPTION
EQUATION
DIAGRAM
FULL
DESCRIPTION
EQUATION
DIAGRAM
FULL SUBTRACTOR
DESCRIPTION
EQUATION
DIAGRAM
KARNAUGH MAP
INTRODUCTION
MAPS
RULES
EXAMPLE
QUINE MCCLUSKEY METHOD
RULES
EXAMPLE
FLIP FLOP
BASIC
ANALYSIS
CLOCKED RS
DESCRIPTION
DIAGRAM
EXCITATION TABLE
JK
DIAGRAM
EXCITATION TABLE
TRIGGER
DIAGRAM
EXCITATION TABLE
DATA
DIAGRAM
EXCITATION TABLE
MS
DIAGRAM
SUBTRACTION BY COMPLEMENTATION
MULTIPLICATION
DIVISION
DEFINITIONS
TIMING DIAGRAM
SUM TERM
PRODUCT TERM
SUM OF PRODUCTS
PRODUCT OF SUMS
MINTERM
MAXTERM
NUMBER SYSTEMS
INTRODUCTION
DECIMAL
BINARY
BINARY CODES
  BINARY CODED DECIMAL
    4221
    GRAY
    EXCESS 3
OCTAL
HEXADECIMAL
CONVERSION
  BINARY TO OCTAL
  BINARY TO HEX
  HEX TO OCTAL
EXAMPLE
  ADDERS
    HALF
    FULL
  K MAP
  QUINE MCCLUSKEY
  FLIP FLOP
    RS
    JK
    TRIGGER
    DATA
    MS
  REGISTERS
    PARALLEL
    SERIAL
  COMBINATIONAL CIRCUITS
  SEQUENTIAL CIRCUITS
  BINARY ARITHMETIC
    ADDITION
    SUBTRACTION
    MULTIPLICATION
    DIVISION
  NUMBER SYSTEMS
  CONVERSIONS
    BINARY TO OCTAL
    BINARY TO HEX
    HEX TO OCTAL

CPU
TEST

GATES
BOOLEAN ALGEBRA
DEFINITIONS
RELATIONS
MINIMIZATION
ADDS
FLIP FLOPS
REGISTERS
COMBINATIONAL CIRCUITS
SEQUENTIAL CIRCUITS
COMPUTER STRUCTURES
BINARY ARITHMETIC
NUMBER SYSTEMS
CONVERSIONS
K MAP
LOGIC DESIGN
COMPREHENSIVE
APPENDIX E

SAMPLE TEXTFILE
To reproduce the entire text included in the application was not practical due to its large size; however, some text is presented in the following pages (especially that corresponding to the sample runs shown in appendix F).
THE 'NOT' GATE, ALSO CALLED THE INVERTER IS REPRESENTED BY A GRAPHIC SYMBOL CONSISTING OF A TRIANGLE FOLLOWED BY A SMALL CIRCLE:

```
\ /
/ \      Y ---- 0 ---- Z
/\      /
```

Y IS THE INPUT TO THE INVERTER, WHILE Z IS THE OUTPUT.

FUNCTIONAL DESCRIPTION:

THE OUTPUT Z OF THE INVERTER IS THE COMPLEMENT OF THE INPUT Y. THAT IS TO SAY THE GATE INVERTS THE INPUT SIGNAL.

LOGICAL EQUATION:

THE ALGEBRIC EQUATION DESCRIBING THE INVERTER CIRCUIT IS:

\[ Z = Y' \]

WE SHALL SHOW THE COMPLEMENT OPERATION BY A PRIME AFTER THE OPERAND. (IN SOME BOOKS A BAR OVER THE OPERAND IS USED).

TRUTH TABLE:

THE INVERTER CIRCUIT'S TRUTH TABLE IS PERHAPS THE EASIEST ONE OF ALL TO REMEMBER:

```
Y | Z
---|---
0 | 1
1 | 0
```
ALTHOUGH COMPUTERS ARE DESIGNED IN VARIOUS WAYS, THE FIVE BASIC ELEMENTS FOUND IN ALL SYSTEMS ARE SHOWN IN THE FOLLOWING BLOCK DIAGRAM AND WILL BE DISCUSSED BRIEFLY.

```
INPUT  MEMORY  OUTPUT
<------>  <------>  <------>
SYSTEM  UNIT  SYSTEM
```

THE "INPUT" SYSTEM TRANSLATES THE DATA PREPARED BY THE USER INTO BINARY INFORMATION WHICH IS MEANINGFUL TO THE DIGITAL CIRCUITRY USED IN THE COMPUTER. THE USER MAY ENTER THE INPUT DATA THROUGH A KEYBOARD, Punched cards, magnetic tape etc.

THE "MEMORY" SYSTEM STORES THE INFORMATION RECEIVED FROM THE INPUT SYSTEM UNTIL IT IS NEEDED. DEVICES USED AS MEMORY ELEMENTS INCLUDE MAGNETIC CORES, MAGNETIC DRUMS AND DISKS, MAGNETIC TAPES AND FLIP FLOPS.

THE "CONTROL" UNIT IS THE "BRAIN" OF THE COMPUTER. IT SELECTS INFORMATION FROM THE MEMORY IN THE PROPER SEQUENCE, INTERPRETS IT AND ACCORDINGLY SENDS CONTROL SIGNALS TO ALL OTHER UNITS.

THE "ARITHMETIC LOGIC UNIT" (ALU) RECEIVES INFORMATION AND COMMANDS FROM THE CONTROL UNIT AND OPERATES ON THE INFORMATION ACCORDINGLY. THE RESULTS ARE THEN PASSED BACK TO THE CONTROL UNIT. USUALLY THE ALU AND THE CONTROL UNIT ARE COMBINED INTO A SINGLE UNIT CALLED THE CENTRAL PROCESSOR UNIT (CPU).

THE "OUTPUT" SYSTEM IS NECESSARY SO THAT THE COMPUTER MAY COMMUNICATE THE RESULTS TO THE USER. THE CONTROL UNIT DECIDES WHAT INFORMATION IS TO BE RETRIEVED FROM THE MEMORY AND SENT TO THE OUTPUT SYSTEM. THE OUTPUT MAY BE DISPLAYED ON A VIDEO TERMINAL, PAPER TELETYPE, LINE PRINTER OR MAY BE STORED ON Punched cards or MAGNETIC TAPE.
IF YOU DO NOT WISH TO CONTINUE WITH A DETAILED DISCUSSION OF THESE TOPICS
TYPE 'QUIT'. YOU MAY REENTER THE DISCUSSION AT THIS POINT LATER BY TYPING
'LEARN COMPUTER STRUCTURES, DETAILS'.

/END
THE COMPUTER AND THE USER MUST BE ABLE TO COMMUNICATE WITH EACH OTHER. THE SECTIONS OF THE COMPUTER WHICH DEAL WITH THIS ARE CALLED INPUT-OUTPUT (I/O) UNITS. BECAUSE THE MACHINES USED FOR I/O (FOR EXAMPLE TELETYPES, LINE-PRINTERS, PUNCHED CARD READERS, MAGNETIC TAPE DRIVES ETC) ARE ELECTROMECHANICAL, they are much slower than the electronic circuitry used in the computer itself. This means that while a user is entering data or is viewing output the central processor unit (CPU) will be sitting idle most of the time. This would be a great waste of valuable computer time. To get around this problem "BUFFER MEMORIES" are used between the I/O units and the main memory as shown below:

```
+-------------------+  +-------------------+
| INPUT             |  | BUFFER            |
| OUTPUT <--- >      |  | MAIN              |
| SYSTEM <--- >      |  | MEMORY <--- >     |
|                   |  |                   |
```

The data being entered by the user will be stored in the buffer memory while the CPU is busy working on some other task. When the buffer memory is full, the CPU will move it to the main memory (which will be very fast) and then resume whatever task it was performing. Usually a smaller processor manages the communication between the I/O equipment and the buffer memory.

The same applies when the computer has to output data to the user. The information is moved from the main memory to the buffer memory by the CPU at a high speed. Then the CPU starts working on something else while the data is being output.

This technique is called "BUFFERED I/O" and is an important part of the "TIME SHARING" concept.

The equipment used for input and output includes the following:

- **Teletypes**: Paper or video terminals using a keyboard.
- **Punched cards**: The standard Hollerith card is 3-1/4 inches by 7-7/8 inches and has 80 columns with 12 rows. A single character can be punched as a 2 or 3 hole code in each column. These cards are then read in by a card reader machine which stores them in the input buffer memory. For output on punched cards a card punching machine is used. The Hollerith code is available under "LEARN HOLLERITH CODE".
- **Punched tape**: A paper tape is similar to a punched card except there are an unlimited number of columns. Each column may contain 7 or more rows and represents one character. At least one row is used to store a timing pulse. The tape is usually 7/8 inch wide and records 10 characters per inch.
Magnetic Tape:
Magnetic tape is the most widely used medium for mass storage. A reel of 1/2 inch wide, 12 inch diameter can contain up to 2400 feet of tape and store millions of characters. Transfer rates are from 20,000 characters per second to many times higher.

Memory:
Any bistable device capable of being set in one of its states, maintaining that state, and indicating that state when interrogated can be used as a memory element. A memory unit is a collection of such memory elements together with associated circuits needed to transfer information in and out of the elements. A memory unit is specified by the number of words it contains and the number of bits in a word.

Computer memories may be classified in various ways. From the structure point of view, there are two types of memories:
1 - "primary" memory: that which is an internal part of the computer and may be accessed directly by the CPU at any time. Examples are magnetic core, flip flops, bubble memories, capacitive memories etc.
2 - "secondary" or "auxiliary" memory: which is used for mass storage external to the system. This memory may be accessed by the CPU through the input-output system. Examples are magnetic tape, magnetic drums and disks.

There are two methods by which memory locations may be addressed:
1 - "random access": internal (primary) memories are random access memories because the CPU can retrieve or store information in any memory location at random, independent of the last memory location addressed.
2 - "sequential access": external memories are usually sequential, meaning that to access a particular memory location it is necessary to start at the beginning and go through the entire memory until the required address is reached.

Memories may fall into either of the following categories:
1 - "static memory": once information has been stored in a flip flop memory it remains there until it is changed. Such a memory is called a static memory.
2 - "dynamic memory": certain memories use capacitive cells to store the information. Because capacitors have certain leakage and cannot retain their charge indefinitely, to retain information in such memories it is necessary to "refresh" the capacitive cells periodically, hence the term "dynamic".

Memories may also be classified as volatile or non-volatile:
A volatile memory (for example flip flops) loses its information when the power is turned off.
POWER IS TURNED OFF.
A NON-VOLATILE MEMORY (FOR EXAMPLE MAGNETIC CORE) RETAINS ITS INFORMATION INDEPENDENT OF THE POWER.

STEXT THE ACCESS TIME IS DEFINED AS THE AVERAGE TIME TO ACCESS A MEMORY LOCATION.
INTERNAL MEMORIES HAVE A SMALLER ACCESS TIME COMPARED TO EXTERNAL MEMORIES.
THE COST PER BIT IS AN IMPORTANT CONSIDERATION. USUALLY THE COST PER BIT IS INVERSELY PROPORTIONAL TO THE ACCESS TIME.
A magnetic core is a tiny doughnut shaped object made of ferromagnetic material. If a wire is passed through the core and a sufficiently large current is passed through it, the core becomes magnetized. The direction of the current determines the polarity of the magnetic field. A positive core (magnetized clockwise) has a value of 1, a negative core (magnetized counterclockwise) has a value of 0. Cores are arranged in a two dimensional matrix for convenient addressing.

If two wires are passed through a core and half the amount of current necessary to magnetize the core is sent through each wire in the proper direction, the core will become magnetized. As shown below the cores are arranged with one core on each intersection so that when current is sent through one horizontal (X) wire and one vertical (Y) wire only one of the cores receives full current and is magnetized. All other cores receive only half the necessary current and remain unaffected.

```
  X0 ---0---0---0---
  X1 ---0---0---0---
  X2 ----0---0---0---
  X3 ------0---0---0---

  Y0    Y1    Y2    Y3
```

A core with two half currents is essentially an "AND" circuit, requiring half currents be applied to both x and y lines in the same direction to change the state of the core. The matrix arrangement, known as the "coincident-current" memory requires only 2n wires to address a matrix of n^2 cores.

Example:

To store a 1 in the core at the intersection of wires X2 and Y3 half currents must be sent through both of these wires. All cores on X2 will receive half the current necessary to magnetize them, all the cores on Y3 will also receive half the necessary currents. Only the core at the intersection of these two wires will receive the full amount of current and will be magnetized, thus storing a 1. To store a 0 in a core, reverse current is used.

/Special

Sensing (reading or retrieving) the data!

/End
To sense the state or value of a certain core the technique used is to pass a common 'sense wire' through each core in the matrix (performing an 'or' function on all the cores). When it is desired to read the state of a core, reverse currents are applied to the appropriate X and Y wires causing the core to 'reset' to 0. If the core was already 0 then no signal appears on the sense line but if the core was a 1 then a current is induced in the sense wire indicating the core contained 1. This however destroys the information from the core and it must be rewritten if it is to be used again.

The inhibit line:

In most applications a fourth line, called the inhibit line is passed through each core in the matrix. Before anything can be written into a memory location it is necessary to read it so that the core is 'reset' to 0. Then if a 1 is to be written, half currents are applied to the appropriate X and Y wires. However if a 0 is to be stored then half currents are still applied to the same X and Y wires but the inhibit wire is applied negative half current, making the total current at the intersection of the X and Y wires equal to half of what is necessary to magnetize the core. This way nothing is written into the core leaving it at 0.

Organization into planes:

Usually in computer applications multi-bit 'words' are used. In a system which uses a n-bit word, the memory is composed of n planes of matrices similar to what have been described above. When a certain address in the memory is addressed, actually n bits, one in each plane are addressed, thus addressing a whole 'word'. Each plane has its own sense wire and inhibit wire.

Auxiliary (secondary) memory:

The most common auxiliary memory devices used in computer systems are magnetic drums, magnetic disks and magnetic tapes. The important characteristics of any device are its access mode, access time, transfer rate, capacity and cost. The average time required to reach a storage location in memory for reading or writing is called the 'access time'. In electromechanical devices with moving parts such as drums, disks and tapes the access time consists of a 'seek' time required to position the read-write head to a location and a 'transfer' time required to transfer data to or from the device.

Magnetic drums and disks are coated with a recording medium. Bits are recorded as magnetic spots on the surface as it passes a stationary 'write' or 'recording head'. Disks are similar to drums except that a flat surface is used instead of a cylinder. Bits are stored on the surface in circular 'tracks'. Each track is divided into 'sectors'. At one time the minimum quantity of information which can be transferred is a sector. At least one track is set aside for timing bits.
THE MAGNETIC TAPE IS A STRIP OF PLASTIC COATED WITH A MAGNETIC RECORDING MEDIUM SUCH AS IRON-OXIDE. BITS ARE RECORDED AS MAGNETIC SPOTS ON THE TAPE ALONG SEVERAL 'TRACKS'. USUALLY 7 OR 9 BITS ARE RECORDED SIMULTANEOUSLY ON PARALLEL TRACKS TO FORM A 'WORD'. HEADS ARE MOUNTED ON EACH TRACK TO MAKE THIS POSSIBLE. THE ACCESS TIME FOR MAGNETIC TAPE IS VERY MUCH LARGER COMPARED TO MAGNETIC DISK AND DRUMS.
THE CPU!

THE CENTRAL PROCESSOR UNIT ACCEPTS INSTRUCTIONS FROM EITHER THE INPUT SYSTEM OR THE MEMORY UNIT AND GENERATES TIMING SIGNALS NECESSARY TO IMPLEMENT THESE INSTRUCTIONS. EACH COMPUTER IS CONSTRUCTED TO EXECUTE A FIXED NUMBER OF "INSTRUCTIONS." THE GATING AND TIMING CIRCUITS NECESSARY TO CARRY THIS OUT CONSTITUTE THE CPU. AMONG OTHER THINGS THE CPU CONSISTS OF THE FOLLOWING "REGISTERS":

1. THE PROGRAM COUNTER (PC)
2. MEMORY ADDRESS REGISTER (MAR)
3. MEMORY BUFFER REGISTER (MBR)
4. OPERATION CODE REGISTER (OCR)
5. ACCUMULATOR (ACC) USED FOR ARITHMETIC OPERATIONS.

THE BLOCK DIAGRAM OF A CPU ALONG WITH THE MEMORY UNIT IS SHOWN BELOW:

[Diagram showing input, system, memory, buffer, operation, code, register, accumulator]

THESE REGISTERS WILL BE DESCRIBED SHORTLY.

THE USER'S INSTRUCTIONS (CALLED A "PROGRAM") MUST BE ENTERED INTO THE COMPUTER'S MEMORY BEFORE THE CPU CAN OPERATE ON THEM. EACH INSTRUCTION CONTAINS TWO FIELDS:

1. THE OPERATION CODE WHICH TELLS THE COMPUTER WHAT TO DO.
2. THE OPERAND'S ADDRESS WHICH TELLS THE COMPUTER WHERE IN THE MEMORY TO FIND THE DATA FOR THE INSTRUCTION.

THE FORMAT OF AN INSTRUCTION IS:

[Diagram showing op-code and address]

ONE INSTRUCTION WORD MAY CONSIST OF A NUMBER OF BITS.

INSTRUCTIONS ARE STORED IN THE MEMORY IN CONSECUTIVE LOCATIONS AS ENTERED BY THE USER. THE CPU FETCHES INSTRUCTIONS FROM CONSECUTIVE MEMORY LOCATIONS.
(STARTING AT LOCATION 0) AND IMPLEMENTS THEM ACCORDING TO THE FOLLOWING ALGORITHM:

INITIALIZE PC ← 0
REPEAT
BEGIN FETCH CYCLE
FETCH THE INSTRUCTION AT LOCATION POINTED TO BY PC AND MOVE IT INTO THE MEMORY ADDRESS REGISTER.
MOVE THE OPERATION CODE PART OF THE INSTRUCTION INTO THE OPERATION CODE REGISTER
INCREMENT THE PROGRAM COUNTER! PC ← PC + 1.
END FETCH CYCLE
BEGIN EXECUTION CYCLE
BECOME THE OPERATION CODE.
SEND OUT THE APPROPRIATE CONTROL SIGNALS TO IMPLEMENT THE INSTRUCTION.
END EXECUTION CYCLE
UNTIL (INSTRUCTION = HALT) OR (THERE IS A RUNTIME ERROR).

THE BASIC INSTRUCTIONS WHICH ARE FOUND IN EVERY COMPUTER ARE:

1 - STORE DATA AT A GIVEN MEMORY LOCATION.
2 - RETRIEVE THE DATA FROM A GIVEN MEMORY LOCATION.
3 - ADD THE CONTENTS OF A GIVEN MEMORY LOCATION TO THE ACCUMULATOR.
4 - SUBTRACT THE CONTENTS OF A GIVEN MEMORY LOCATION FROM THE ACCUMULATOR.
5 - JUMP (FORCE THE PC TO POINT) TO A GIVEN MEMORY LOCATION.
6 - HALT.
7 - READ A WORD OF DATA FROM THE INPUT SYSTEM INTO THE ACCUMULATOR.
8 - WRITE A WORD OF DATA FROM THE ACCUMULATOR TO THE OUTPUT SYSTEM.

THE PROGRAM COUNTER IS USED TO PROVIDE A WAY TO PROPERLY SEQUENCE THE FETCHING OF INSTRUCTIONS STORED IN MEMORY.
INITIALLY THE PC CONTAINS 0 AND WILL FETCH THE FIRST INSTRUCTION FROM MEMORY LOCATION 0. AFTER THIS THE PC IS INCREMENTED BY 1 EACH TIME AND POINTS TO THE NEXT SEQUENTIAL LOCATION UNLESS THE PREVIOUS INSTRUCTION WAS A "JUMP" INSTRUCTION WHICH CAN "FORCE" THE PC TO BE SET TO AN ADDRESS SPECIFIED IN THE INSTRUCTION ITSELF.

THE MEMORY BUFFER REGISTER IS A "BUFFER" REGISTER USED TO STORE INFORMATION TEMPORARILY AFTER IT HAS BEEN RETRIEVED FROM THE MEMORY, OR JUST BEFORE IT IS STORED IN THE MEMORY.

THE OPERATION CODE REGISTER!
WHEN THE CPU FETCHES AN INSTRUCTION FROM THE MEMORY IT IS INITIALLY MOVED INTO THE MBR. FROM THERE THE OPERATION CODE PART OF THE INSTRUCTION IS MOVED INTO THE OCR WHERE IT IS DECODED.

THE MEMORY ADDRESS REGISTER!
WHEN THE CPU RECEIVES AN INSTRUCTION TO STORE OR RETRIEVE A DATA WORD FROM THE MEMORY, THE ADDRESS OF THE MEMORY LOCATION IS SPECIFIED IN THE INSTRUCTION.
THIS ADDRESS IS MOVED FROM THE HDR TO THE HAR AND THUS SENDS THE APPROPRIATE SIGNALS TO REFERENCE THE DESIRED LOCATION. THEN THE DATA IS MOVED FROM THE ACCUMULATOR TO THE HDR, AND THEN TO THE MEMORY. THE SAME PROCESS OCCURS WHEN DATA IS TO BE RETRIEVED FROM THE MEMORY. ONLY INFORMATION IS MOVED FROM THE MEMORY TO THE HDR AND THEN TO THE ACCUMULATOR.

THE ACCUMULATOR:

THE ACCUMULATOR IS A GENERAL PURPOSE REGISTER AVAILABLE TO THE USER (PROGRAMMER) TO STORE ONE DATA WORD AND OPERATE ON IT. WHEN AN INSTRUCTION IS FETCHED FROM THE MEMORY, THE OPERAND (IF ANY) IS MOVED FROM THE HDR INTO THE ACCUMULATOR. IN LARGER MACHINES THERE MAY BE MORE THAN ONE ACCUMULATOR AND IT IS NECESSARY TO SPECIFY A PARTICULAR ACCUMULATOR IN THE INSTRUCTION.

ONCE AN INSTRUCTION HAS BEEN DECODED IF IT IS AN ARITHMETIC INSTRUCTION THEN APPROPRIATE SIGNALS ARE SENT TO THE ARITHMETIC UNIT WHICH OPERATES ON THE DATA IN THE ACCUMULATOR AND SOME DATA IN THE MEMORY LOCATION SPECIFIED BY THE HAR.

THE ARITHMETIC UNIT MAY CONSIST OF THE FOLLOWING CIRCUITRY:

1 - ADDER (ALSO USED FOR SUBTRACTION)
2 - MULTIPLIER
3 - DIVIDER
4 - CIRCUITS TO PERFORM LOGICAL OPERATIONS (COMPLEMENTING, AND-ING, OR-ING, SHIFTING BITS LEFT OR RIGHT ETC.).
CONSIDER THE LOGIC DIAGRAM SHOWN BELOW. YOU ARE TO WRITE A LOGIC EQUATION WHICH WILL DESCRIBE THE OUTPUT (F) OF THIS CIRCUIT IN TERMS OF THE INPUTS X, Y, AND Z.

\[ F = X'Y + Z + X'Y \]

FIGURE 1

WHAT WILL BE THE EXPRESSION FOR THE SIGNAL AT POINT P2?
\[ X'Y \]

WHAT WILL BE THE EXPRESSION AT POINT P3?
\[ Z + X'Y \]

WHAT WILL BE THE EXPRESSION AT F?
\[ X'Y + Z + X'Y \]

CAN THE EXPRESSION YOU TYPED BE MINIMIZED?
YES

IF YOU WERE ASKED TO DRAW A LOGIC DIAGRAM TO REPRESENT THIS MINIMIZED FUNCTION, HOW MANY GATES IN ALL WOULD YOU NEED?
3
OK, HERE ARE THE THREE GATES YOU WILL NEED!

\[ \text{FIGURE 2} \]

LET'S SEE IF YOU KNOW HOW TO CONNECT THEM TOGETHER TO GET \( X \cdot Y + Z \) AT TERMINAL 03.

AT WHICH TERMINAL IN FIGURE 2 WILL YOU APPLY THE INPUT \( Y \)?

\( \text{RIGHT01} \ #1 \)
\( \text{RIGHT01} \ #2 \)
\( \text{TEXT} \)

AND WHERE WILL YOU APPLY INPUT \( Z \)?

\( \text{RIGHT01} \ #1 \)
\( \text{RIGHT01} \ #2 \)
\( \text{TEXT} \)

WELL, YOU MADE IT THROUGH THIS QUIZ! BUT IT WASN'T THAT HARD, WAS IT?

\( ^* \END \)
APPENDIX F

SAMPLE RUNS
Hi! Seems like this is the first time we are meeting! Will you please type in your first name? Tahir. Your last name? Hufti.

Well, Tahir. Glad to meet you!

If you want to find out how to talk with me, type help!

Waiting for your instructions....

>> help
You have the following options:

- Type: learn <subject> to get information about <subject>
- Example: <subject> to see some examples about <subject>
- Test <subject> to go through a quiz on <subject>
- Comments to enter comments about this CAI system

You may use the following monitor commands after you exit from this program:

- Print Direct.CAI to get a listing of the subjects available
- Print Help.CAI to get a copy of the detailed instructions

Type: More Help if you need it.

Waiting for your instructions....

>> learn gates
Want to look at a list of what's available under 'learn, gates'? Yes

- Introduction
- AND
- OR
- Inverter
- NAND
- NOR
- Exclusive OR
- Equivalence

What would you like to see under 'learn, gates'? >> Inverter
Want to look at a list of what's available under 'learn, gates, inverter'? Yes

Diagram
THE INVERTER:

THE 'NOT' GATE, also called the inverter is represented by a graphic symbol consisting of a triangle followed by a small circle:

```
\ /  \
Y ----| 0----Z
\ /  \\
```

Y is the input to the inverter, while Z is the output.

FUNCTIONAL DESCRIPTION:

The output Z of the inverter is the complement of the input Y, that is to say the gate 'inverts' the input signal.

LOGICAL EQUATION:

The algebraic equation describing the inverter circuit is:

\[ Z = Y' \]

We shall show the complement operation by a prime after the operand. (In some books a bar over the operand is used).

TRUTH TABLE:

The inverter circuit's truth table is perhaps the easiest one of all to remember!
Y  Z
---------
0 1 1
1 1 0

REQUESTING FOR YOUR INSTRUCTIONS ....

EXAMPLE
CAN GET A LINE-PRINTER COPY OF THE DIRECTORY LISTING FOR 'EXAMPLE'
TYPEING THE MONITOR COMMAND:
.PRINT DIRECT.CAI
IF YOU EXIT FROM THIS PROGRAM,
I TO GO AHEAD AND LOOK AT A LIST OF WHAT'S AVAILABLE UNDER 'EXAMPLE'
THIS TERMINAL? NO

I WOULD YOU LIKE TO SEE UNDER 'EXAMPLE'?
NOTHING

REQUESTING FOR YOUR INSTRUCTIONS ....

EXAMPLE QUINE MCCLUSKEY
I TO LOOK AT A LIST OF WHAT'S AVAILABLE UNDER 'EXAMPLE, QUINE MCCLUSKEY'? 

MORE ON QUINE-MCCLUSKEY METHOD FOR MINIMIZATION OF BOOLEAN FUNCTIONS:
QUINE MCCLUSKEY METHOD WILL BE USED TO MINIMIZE THE FUNCTION:
F (A, B, C) = MINTERMS(2,4,6,8,9,10,12,13,15)

STEP 1 GROUP THE BINARY REPRESENTATIONS OF THE MINTERMS ACCORDING TO THE 
BIT OF 1 BITS. AS SHOWN BELOW, THERE ARE FOUR GROUPS. GROUP 1 INCLUDES
SE MINTERMS WHICH CONTAIN ONLY ONE 1 BIT, GROUP 2 INCLUDES THOSE TERMS WHICH
TAIN TWO 1 BITS, AND SO ON:

MINTERM ABCD
---------------------
2 0010
4 0100 GROUP 1 (ONE 1 BIT)
0 1000
---------------------
STEP 2 - NOW WE HAVE TO COMBINE TERMS WHICH DIFFER IN ONLY ONE BIT. IF YOU LOOK CAREFULLY AT THE TABLE ABOVE, YOU WILL SEE THAT WE CAN COMBINE TERMS IN GROUP 1 ONLY WITH THOSE IN GROUP 2. IT IS NOT POSSIBLE TO COMBINE TWO TERMS WHICH ARE IN THE SAME GROUP.

FIRST WE LOOK AT MINTERM 2 (WHICH IS THE FIRST TERM IN GROUP 1) AND TRY TO COMBINE IT WITH THE TERMS OF GROUP 2. WHEN 0010 IS COMPARED WITH 0110 WE SEE THAT THE TWO TERMS DIFFER ONLY IN THE THIRD BIT FROM THE RIGHT. SO WE COMBINE THE TWO TERMS BY PLACING A 'DASH' IN THIS POSITION, GETTING THE TERM: 0-10.

NEXT WE COMPARE THE SAME TERM FROM GROUP 1 (0010) WITH THE NEXT TERM IN GROUP 2 (1001). BECAUSE THESE TERMS DIFFER IN TWO BITS (THE FIRST AND THE FOURTH FROM THE RIGHT) SO WE CANNOT COMBINE THEM.

NEXT WE COMPARE THE SAME TERM FROM GROUP 1 WITH THE NEXT TERM IN GROUP 2 (1010) AND SEE THAT THE TWO TERMS DIFFER ONLY IN THE FOURTH BIT FROM THE RIGHT, SO WE CAN COMBINE THEM, PLACING A DASH IN THE POSITION WHERE THE BITS ARE DIFFERENT, GETTING: 0-10.

NEXT THE SAME TERM FROM GROUP 1 IS COMPARED WITH THE NEXT TERM IN GROUP 2 (1100). WE SEE THAT THERE ARE TWO BITS WHICH DO NOT MATCH SO THESE TWO TERMS CANNOT BE COMBINED.

NOW BECAUSE THE FIRST TERM OF GROUP 1 HAS BEEN COMPARED WITH ALL THE TERMS IN THE SECOND GROUP, WE REPEAT THE PROCESS WITH THE NEXT TERM FROM GROUP 1 UNTIL ALL TERMS IN THE TWO GROUPS HAVE BEEN COMPARED.


THE RESULT OF THIS PROCESS ARE SHOWN IN TABLE # 2 BELOW. THE LEFT COLUMN OF LIST # 2 SHOWS WHICH MINTERMS WERE COMBINED TO THE TERM SHOWN IN THE RIGHT COLUMN.

<table>
<thead>
<tr>
<th>LIST # 1</th>
<th>LIST # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINTERM</td>
<td>ABCD</td>
</tr>
<tr>
<td>2 0010</td>
<td>2, 6</td>
</tr>
<tr>
<td>4 0100</td>
<td>4, 6</td>
</tr>
<tr>
<td>8 1000</td>
<td>8, 9</td>
</tr>
<tr>
<td>12 1100</td>
<td>8-12</td>
</tr>
<tr>
<td>6 0110</td>
<td>9, 10</td>
</tr>
</tbody>
</table>
The asterisks next to the terms in List #1 are placed there to show that it was possible to combine that term with at least one term of the adjacent group. Sometimes there are certain terms which cannot be combined with others, as you will see below.

You can see List #2 is also divided into groups according to the number of 1 bits in the terms.

We now repeat the whole process with List #2. Terms which differ in only one bit will be combined together. The first term of the first group in List #2 (0-10) cannot be combined with any term of the second group. Similarly, the second, third and fourth terms cannot be combined either. However, the fifth term (100-) can be combined with the second term of the second group (110-) because the two terms differ only in the third bit from the right, giving 1-0-.

Next we see the term 1-00 from Group 1 can be combined with term 1-01 from Group 2 giving the minimized term: 1-0-.

These are all the terms that can be combined, so List #3 is very short:

<table>
<thead>
<tr>
<th>List #2</th>
<th>List #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minterms ABCD</td>
<td>Minterms ABCD</td>
</tr>
<tr>
<td>2+6 0-10</td>
<td>8,9,12,13 1-0-</td>
</tr>
<tr>
<td>2+10 0-10</td>
<td></td>
</tr>
<tr>
<td>4+6 01-0</td>
<td></td>
</tr>
<tr>
<td>4+12 10-0</td>
<td></td>
</tr>
<tr>
<td>8,9 100- *</td>
<td></td>
</tr>
<tr>
<td>8,10 10-0</td>
<td></td>
</tr>
<tr>
<td>8,12 1-00 *</td>
<td></td>
</tr>
</tbody>
</table>

You will see that there is only one term in List #3, the reason being that when we combine the term for minterms 8+9 (Group 1 in List #2) with minterms 12+13 (Group 2, List #2), the resulting term is 1-0- (which corresponds to minterms 8+9 and 12+13).

Also, when we combine the term for minterms 8+12 (Group 1 in List #2) with the term for minterm 9+13 (Group 2 List #2) we get the term 1-0- which also corresponds to minterms 8+9+12 and 13. As both of these terms are the same, one can be dropped.

The asterisks in List #2 show the terms which it was possible to combine with other terms, the rest of the terms could not be combined, so they are "Prime Implicants". Also, the single term in List #3 cannot be combined with any other term so it is also a Prime Implicant.
STEP 3 - TO DETERMINE THE SMALLEST NUMBER OF PRIME IMPLICANTS WE FORM A
"PRIME IMPLICANT CHART". THE PRIME IMPLICANTS ARE LISTED VERTICALLY, ALL THE
MINTERMS FOR THE GIVEN FUNCTION ARE LISTED HORIZONTALLY. AN "X" IS PLACED IF A
MINTERM IS COVERED BY A PRIME IMPLICANT:

<table>
<thead>
<tr>
<th>MINTERMS</th>
<th>1 2 4 6 8 9 10 12 13 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIME IMPLICANTS</td>
<td>___</td>
</tr>
<tr>
<td>1,2,12,13</td>
<td>1-0-</td>
</tr>
<tr>
<td>2,6</td>
<td>0-10</td>
</tr>
<tr>
<td>2,10</td>
<td>-010</td>
</tr>
<tr>
<td>4,6</td>
<td>01-0</td>
</tr>
<tr>
<td>4,12</td>
<td>-100</td>
</tr>
<tr>
<td>8,10</td>
<td>10-0</td>
</tr>
<tr>
<td>13,15</td>
<td>11-1</td>
</tr>
</tbody>
</table>

LOOKING AT THE COLUMNS FOR MINTERMS 9 AND 15 WE SEE THAT THEY ARE COVERED BY
ONLY ONE "PRIME IMPLICANT (ONLY ONE "X")", THIS MEANS THAT THE CORRESPONDING
PRIME IMPLICANTS ARE "ESSENTIAL" PRIME IMPLICANTS, AND MUST APPEAR IN THE FINAL
MINIMIZED EXPRESSION.

NOTE THAT ONCE WE CHOOSE THESE TWO ESSENTIAL PRIME IMPLICANTS, THEY WILL
ALSO COVER MINTERMS 8, 9, 12 AND 13. ALL THE MINTERMS WHICH ARE COVERED ARE
NOW DROPPED FROM THE TABLE, AND WE GET A MUCH SMALLER TABLE!

<table>
<thead>
<tr>
<th>MINTERMS</th>
<th>2 4 6 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMES</td>
<td>____</td>
</tr>
<tr>
<td>2,6</td>
<td>0-10</td>
</tr>
<tr>
<td>2,10</td>
<td>-010</td>
</tr>
<tr>
<td>4,6</td>
<td>01-0</td>
</tr>
<tr>
<td>4,12</td>
<td>-100</td>
</tr>
<tr>
<td>8,10</td>
<td>10-0</td>
</tr>
</tbody>
</table>

NOW THE OBJECTIVE IS TO COVER MINTERMS 2, 4, 6 AND 10 WITH A MINIMUM NUMBER
OF PRIME IMPLICANTS (WE HAVE FIVE TO CHOOSE FROM). THIS CAN BE DONE BY TAKING
PRIME IMPLICANTS -010 AND 01-0.

COMBINING THESE TWO PRIME IMPLICANTS WITH THE TWO ESSENTIAL PRIME IMPLICANTS
(1-0- AND -010), THE MINIMIZED FUNCTION CAN BE EXPRESSED AS:

\[
F(A\cdot B\cdot C\cdot D) = 1\cdot 0\cdot - + 0\cdot 1\cdot 0 + 0\cdot 1\cdot - + 1\cdot 1\cdot -
\]

(Replacing the 1's by the corresponding variable, and 0's by the
corresponding complemented variable).

THIS IS THE MINIMIZED FORM OF THE GIVEN FUNCTION

\[
F(A\cdot B\cdot C\cdot D) = \text{MINTERMS (2,4,6,8,9,10,12,13,15)}
\]
RUN LEARN

III! TAHIR. NICE TO SEE YOU AGAIN!

DO YOU WANT TO LOOK AT YOUR PAST RECORD? NO

IF YOU FORGOT HOW TO TALK WITH ME, ASK ME FOR HELP!

WAITING FOR YOUR INSTRUCTIONS....

>> LEARN COMPUTER STRUCTURE

WANT TO LOOK AT A LIST OF WHAT'S AVAILABLE UNDER 'LEARN, COMPUTER STRUCTURE'? YES

INTRODUCTION

BASIC DETAILS

WHAT WOULD YOU LIKE TO SEE UNDER 'LEARN, COMPUTER STRUCTURE'? YES

>> INTRODUCTION

ALTHOUGH COMPUTERS ARE DESIGNED IN VARIOUS WAYS, THE FIVE BASIC ELEMENTS FOUND IN ALL SYSTEMS ARE SHOWN IN THE FOLLOWING BLOCK DIAGRAM AND WILL BE DISCUSSED BRIEFLY.

```
INPUT            MEMORY            OUTPUT
<------------->   <----->            <------>
SYSTEM           UNIT               SYSTEM
<-------->        <----->            <----->

CONTROL <----- ARITHMETIC <----->

UNIT             UNIT
<-------->        <----->
```

WAITING FOR YOUR INSTRUCTIONS....
MAGNETIC CORE MEMORY:
(Magnetized clockwise) has a value of 1. A negative core (magnetized counterclockwise) has a value of 0. Cores are arranged in a two-dimensional matrix for convenient addressing.

If two wires are passed through a core and half the amount of current necessary to magnetize the core is sent through each wire in the proper direction, the core will become magnetized. As shown below, the cores are arranged with one core on each intersection so that when current is sent through one horizontal (X) wire and one vertical (Y) wire, only one of the cores receives full current and is magnetized. All other cores receive only half the necessary current and remain unaffected.

A core with two half currents is essentially an "AND" circuit, requiring half currents be applied to both X and Y lines in the same direction to change the state of the core. The matrix arrangement, known as the "Coincident-Current" memory requires only 2N wires to address a matrix of N"2 cores.

Example:
To store a 1 in the core at the intersection of wires X2 and Y3, half currents must be sent through both of these wires. All cores on X2 will receive half the current necessary to magnetize them. All the cores on Y3 will also receive half the necessary currents. Only the core at the intersection of these two wires will receive the full amount of current and will be magnetized, thus storing a 1. To store a 0 in a core, reverse current are used.

Sensing (reading or retrieving) the data:
To sense the state or value of a certain core the technique used is to pass a common "sense wire" through each core in the matrix (performing an "OR" function on all the cores). When it is desired to read the state of a core, reverse currents are applied to the appropriate X and Y wires causing the core to "reset" to 0. If the core was already 0 then no signal appears on the sense line. But if the core was a 1 then a current is induced in the sense wire indicating the core contained 1. This however destroys the information from
THE CORE AND IT MUST BE REWRITTEN IF IT IS TO BE USED AGAIN.

THE INHIBIT LINE:
In most applications a fourth line, called the inhibit line, is passed through each core in the matrix. Before anything can be written into a memory location it is necessary to read it so that the core is "reset" to 0. Then if a 1 is to be written, half currents are applied to the appropriate X and Y wires. However if a 0 is to be stored then half currents are still applied to the same X and Y wires but the inhibit wire is applied negative half current, making the total current at the intersection of the X and Y wires equal to half of what is necessary to magnetize the core. This way nothing is written into the core leaving it at 0.

ORGANIZATION INTO PLANES:
Usually in computer applications multi-bit "words" are used. In a system which uses a n-bit word, the memory is composed of n planes of matrices similar to what have been described above. When a certain address in the memory is addressed, actually n bits; one in each plane are addressed, thus addressing a whole "word". Each plane has its own sense wire and inhibit wire.

WAITING FOR YOUR INSTRUCTIONS ....

>> QUIT

SEE YOU LATER. GOOD NIGHT !!!

EXIT
RUN LEARN

HI ! TAHIR. NICE TO SEE YOU AGAIN!

DO YOU WANT TO LOOK AT YOUR PAST RECORD? YES
DO YOU WANT TO LOOK AT YOUR ENTIRE RECORD? NO
15-JUL-81 19:56:43 TEST LOGIC DESIGN, 1
QUESTIONS ASKED: 0  WRONG: 2

IF YOU FORGOT HOW TO TALK WITH ME, ASK ME FOR HELP!

WAITING FOR YOUR INSTRUCTIONS ....

>> TEST K MAP; 1

### PLEASE READ THESE INSTRUCTIONS CAREFULLY. IF YOU DO NOT FOLLOW THEM, YOU MIGHT LOSE POINTS EVEN IF YOUR ANSWER IS CORRECT!
1 - WHEN AN ANSWER CALLS FOR A NUMBER, TYPE DECIMAL DIGITS.
   FOR EXAMPLE: TYPE 5, NOT FIVE.
2 - TYPE YOUR ANSWER ON A SINGLE LINE.
3 - SEPERATE PARTS OF AN ANSWER BY COMMAS. FOR EXAMPLE: 001, 101, 111
4 - IN YOUR ANSWER YOU MAY USE BLANKS FREELY.

YOU ARE TO USE THE KARNAUGH MAP TO MINIMIZE THE FOLLOWING FUNCTION:

\[
F(A'B'C) = A'B'C + A'BC + ABC
\]

HOW MANY SQUARES WILL THIS MAP HAVE? 18
RIGHT!
THE K-MAP YOU WILL USE LOOKS LIKE THIS:

```
\[ \begin{array}{c}
\text{A} \\
\hline
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array} \]
```

Each of the 8 squares must be assigned a minterm. Type in the minterms assigned to the 4 squares in the top row, going from left to right (separated by commas): 0, 1, 3, 2

Correct!

For the bottom row (left --> right): 7, 5, 3, 6

You're doing fine!

So now the K-map looks like this:

```
\[ \begin{array}{c}
\text{A} \\
\hline
0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
\hline
0 & 0 & 1 & 1 & 3, 2 \\
1 & 4, 5 & 51 & 71 & 61 \\
\end{array} \]
```

F(A, B, C) = A'B'C + A'DC + ABC

From now on we will refer to the squares by the corresponding decimal numbers. Type the decimal numbers corresponding to the squares in which you will plot 1's: 1, 5, 7.

That's right!

So now the map looks like this:

```
\[ \begin{array}{c}
\text{A} \\
\hline
0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
\hline
0 & 0 & 1 & 1 & 3, 2 \\
1 & 4, 5 & 51 & 71 & 61 \\
\end{array} \]
```

F(A, B, C) = A'B'C + A'DC + ABC

How many groups of 1's can you form in this map? 3

No, that is not right.

Try again: 2

That's right!

Type in the decimal numbers corresponding to the squares for any one of
THE GROUPS: 1, 3
RIGHT!

NOW TYPE IN THE DECIMAL NUMBERS CORRESPONDING TO THE SQUARES FOR THE SECOND GROUP: 1, 3
YOU HAVE ALREADY ENTERED THIS GROUP! TRY AGAIN. 3, 7
RIGHT!

WHAT PRODUCT TERM WILL YOU WRITE FOR THE GROUP COVERING SQUARES 1 AND 3? A'C
THAT'S RIGHT!
WHAT PRODUCT TERM WILL YOU WRITE FOR THE GROUP COVERING SQUARES 3 AND 7? B'C
RIGHT!

TYPE THE FINAL EXPRESSION FOR THE MINIMIZED FUNCTION:
F(A, B, C) = A'C + B'C
CORRECT!

SEEMS LIKE YOU MADE IT THROUGH THIS QUIZ!!!

> QUESTIONS ASKED: 10  WRONG: 1
.RUN LEARN

HI! Tahir. Nice to see you again!

Do you want to look at your past record? Yes
Do you want to look at your entire record? No

If you forgot how to talk with me, ask me for help!

Waiting for your instructions....

>> TEST LOGIC DESIGN, # 1 /EXPERT

Consider the logic diagram shown below. You are to write a logic equation which will describe the output (F) of this circuit in terms of the inputs X, Y, and Z

\[
\begin{align*}
\text{X} & \quad \text{AND} \quad \text{Y} \quad \text{OR} \quad \text{Z} \\
\text{F} & \quad \text{AND} \quad \text{OR} \\
\end{align*}
\]

FIGURE 1

What will be the expression for the signal at point P2? X'Y

Right!

What will be the expression at point P3? X'YZ

Wrong!

Try again: X'YZ

No, that is not right.

Try again: X'Y+Z

That's right!

What will be the expression at F? X'Y+Z\times Y

You're doing fine!

Can the expression you typed be minimized? No

Oh yes it can!

What will be the minimized expression? X'Y+Z

That's right!

If you were asked to draw a logic diagram to represent this minimized function, how many gates in all would you need? 3
RIGHT!
OK, HERE ARE THE THREE GATES YOU WILL NEED!

\[ \text{FIGURE 2} \]

LETS SEE IF YOU KNOW HOW TO CONNECT THEM TOGETHER TO GET \( X'Y + Z \) AT TERMINAL 03.
AT WHICH TERMINAL IN FIGURE 2 WILL YOU APPLY THE INPUT \( Y' \) ? A2
RIGHT!
AND WHERE WILL YOU APPLY INPUT \( Z \) ? A1
THAT'S NOT CORRECT.
TRY AGAIN! A1
RIGHT!

WELL, YOU MADE IT THROUGH THIS QUIZ! BUT IT WASN'T THAT HARD, WAS IT?

> QUESTIONS ASKED: 8 WRONG: 2
WAITING FOR YOUR INSTRUCTIONS....

>> FINISH
SEE YOU LATER. HAVE A NICE EVENING!
EXIT
BIBLIOGRAPHY


Carrol, J. B. 1963,64. A Model for School Learning. Teachers College Record. (pp. 723-733).


