8-1979

A microprocessor-based mathematics trainer with voice output.

Michael Joseph Linnig 1956-
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

Recommended Citation
https://doi.org/10.18297/etd/838

This Master's Thesis is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.
A MICROPROCESSOR-BASED MATHEMATICS TRAINER WITH VOICE OUTPUT

By

Michael Joseph Linnig
B. S., University of Louisville, 1978

A Thesis
Submitted to the Faculty of the
University of Louisville
Speed Scientific School
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Electrical Engineering

August 1979
A MICROPROCESSOR-BASED MATHEMATIC TRAINER WITH VOICE OUTPUT

Submitted by: Michael Joseph Linnig

A Thesis Approved on

\[ \text{July 30, 1979} \]

by the Following Reading and Examination Committee:

Thesis Director, Barry R. Horowitz

Donald L. Kalmey

Thomas M. Murray, Jr.

Emerson Foulke
ACKNOWLEDGEMENTS

The author would like to thank: Dr. Barry R. Horowitz, who first conceived the Mathematics Trainer, for his help and guidance in this project; Dr. Emerson Foulke and Dr. Charles Thompson for their advice in the areas of human factors and education; and the author's associates in The Perceptual Alternatives Lab, especially, Mr. Brian Garvey, Mr. John Kilpatrick, and Mr. Jimmy Ryan. The author would also like to thank Dr. Peter Aronhime and Mr. Don Fleischer for their help in producing the program listings, and Dr. Donald Kalmey and Professor Thomas Murray for serving on his thesis committee. The author is grateful to the Graduate School for their grant that made the device financially possible. Most of all the author would like to thank his parents for their support and encouragement in the past 23 years.
ABSTRACT

The design and development of a microprocessor-controlled mathematics trainer is described. The trainer is designed primarily for a blind student, and uses synthetic speech to verbally present mathematics problems in the form of incomplete mathematical sentences. The student responds via numeric keypad. Feedback is given to the student immediately after each problem and after each problem session. Several options give the teacher flexibility in pacing the student. In addition, the teacher can create individualized problem sets or choose from many predefined problem sets, arranged in order of difficulty.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROVAL PAGE</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>A. The Problem</td>
<td>1</td>
</tr>
<tr>
<td>B. Project Motivation</td>
<td>1</td>
</tr>
<tr>
<td>C. Some Talking Devices</td>
<td>2</td>
</tr>
<tr>
<td>1. Texas Instruments' &quot;Speak and Spell&quot;</td>
<td>2</td>
</tr>
<tr>
<td>2. The &quot;Speed Plus&quot; Calculator</td>
<td>3</td>
</tr>
<tr>
<td>3. The Votrax</td>
<td>3</td>
</tr>
<tr>
<td>II. FUNCTIONAL SPECIFICATIONS AND MAN-MACHINE INTERFACE</td>
<td>4</td>
</tr>
<tr>
<td>A. Device Overview</td>
<td>4</td>
</tr>
<tr>
<td>B. Problem Generation</td>
<td>7</td>
</tr>
<tr>
<td>1. Range Mode</td>
<td>7</td>
</tr>
<tr>
<td>2. Level of Difficulty</td>
<td>9</td>
</tr>
<tr>
<td>C. Problem Presentation</td>
<td>10</td>
</tr>
<tr>
<td>1. Learning Considerations</td>
<td>10</td>
</tr>
<tr>
<td>a. Format</td>
<td></td>
</tr>
<tr>
<td>b. Pacing Options</td>
<td>15</td>
</tr>
<tr>
<td>2. Man-Machine Considerations</td>
<td>15</td>
</tr>
<tr>
<td>D. Teacher Programming</td>
<td>16</td>
</tr>
<tr>
<td>III. SYSTEM DESIGN</td>
<td>20</td>
</tr>
<tr>
<td>A. System Requirements</td>
<td>20</td>
</tr>
<tr>
<td>B. Subsystem Selection</td>
<td>23</td>
</tr>
<tr>
<td>IV. HARDWARE DESIGN</td>
<td>26</td>
</tr>
<tr>
<td>A. The Development System</td>
<td>26</td>
</tr>
<tr>
<td>1. The Controller</td>
<td>26</td>
</tr>
<tr>
<td>2. Memory Expansion</td>
<td>26</td>
</tr>
<tr>
<td>B. Final System Design</td>
<td>27</td>
</tr>
<tr>
<td>1. The Controller</td>
<td>27</td>
</tr>
<tr>
<td>2. Firmware</td>
<td>30</td>
</tr>
<tr>
<td>3. Interface Circuitry</td>
<td>32</td>
</tr>
<tr>
<td>a. Overview</td>
<td>32</td>
</tr>
<tr>
<td>C. Display and Keyboard Circuitry</td>
<td>35</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Seven Segment Displays</td>
<td>35</td>
</tr>
<tr>
<td>2.</td>
<td>Minus and Equal Signs</td>
<td>35</td>
</tr>
<tr>
<td>3.</td>
<td>Emergency Blanking Circuit</td>
<td>35</td>
</tr>
<tr>
<td>4.</td>
<td>Operation Sign</td>
<td>37</td>
</tr>
<tr>
<td>5.</td>
<td>User Prompts</td>
<td>37</td>
</tr>
<tr>
<td>6.</td>
<td>Interface Summary</td>
<td>41</td>
</tr>
<tr>
<td>V.</td>
<td>SOFTWARE</td>
<td>45</td>
</tr>
<tr>
<td>A.</td>
<td>Software Development</td>
<td>45</td>
</tr>
<tr>
<td>B.</td>
<td>The Software</td>
<td>46</td>
</tr>
<tr>
<td>1.</td>
<td>Control Subroutines</td>
<td>48</td>
</tr>
<tr>
<td>a.</td>
<td>MAIN Program</td>
<td>48</td>
</tr>
<tr>
<td>b.</td>
<td>DRILL Subroutine</td>
<td>53</td>
</tr>
<tr>
<td>c.</td>
<td>PROGRAM Subroutine</td>
<td>53</td>
</tr>
<tr>
<td>d.</td>
<td>REPORT Subroutine</td>
<td>58</td>
</tr>
<tr>
<td>e.</td>
<td>TEACH Subroutine</td>
<td>58</td>
</tr>
<tr>
<td>2.</td>
<td>Input/Output Subroutines</td>
<td>62</td>
</tr>
<tr>
<td>a.</td>
<td>BLANK Subroutine</td>
<td>62</td>
</tr>
<tr>
<td>b.</td>
<td>BLANKIT Subroutine</td>
<td>62</td>
</tr>
<tr>
<td>c.</td>
<td>DISPLAY Subroutine</td>
<td>62</td>
</tr>
<tr>
<td>d.</td>
<td>GETDIS Subroutine</td>
<td>62</td>
</tr>
<tr>
<td>e.</td>
<td>GETEM Subroutine</td>
<td>68</td>
</tr>
<tr>
<td>f.</td>
<td>PACK Subroutine</td>
<td>68</td>
</tr>
<tr>
<td>g.</td>
<td>SAYDIS Subroutine</td>
<td>68</td>
</tr>
<tr>
<td>h.</td>
<td>SCAN Subroutine</td>
<td>68</td>
</tr>
<tr>
<td>i.</td>
<td>SPKEY Subroutine</td>
<td>77</td>
</tr>
<tr>
<td>j.</td>
<td>SPKWD and LDPAUS Subroutines</td>
<td>77</td>
</tr>
<tr>
<td>k.</td>
<td>UNPACK Subroutine</td>
<td>81</td>
</tr>
<tr>
<td>l.</td>
<td>WAIT Subroutine</td>
<td>81</td>
</tr>
<tr>
<td>3.</td>
<td>Arithmetic Subroutines</td>
<td>81</td>
</tr>
<tr>
<td>a.</td>
<td>ADDEM Subroutine</td>
<td>85</td>
</tr>
<tr>
<td>b.</td>
<td>BCDADD Subroutine</td>
<td>85</td>
</tr>
<tr>
<td>c.</td>
<td>BCDINC Subroutine</td>
<td>85</td>
</tr>
<tr>
<td>d.</td>
<td>BCDMULT Subroutine</td>
<td>85</td>
</tr>
<tr>
<td>e.</td>
<td>BISUB Subroutine</td>
<td>85</td>
</tr>
<tr>
<td>f.</td>
<td>COMPARE Subroutine</td>
<td>89</td>
</tr>
<tr>
<td>g.</td>
<td>DADD Subroutine</td>
<td>89</td>
</tr>
<tr>
<td>h.</td>
<td>GETANS Subroutine</td>
<td>89</td>
</tr>
<tr>
<td>i.</td>
<td>HI-LO Subroutine</td>
<td>93</td>
</tr>
<tr>
<td>j.</td>
<td>RANDOM Subroutine</td>
<td>93</td>
</tr>
<tr>
<td>4.</td>
<td>Special Function Subroutines</td>
<td>93</td>
</tr>
<tr>
<td>a.</td>
<td>ERROR Subroutine</td>
<td>97</td>
</tr>
<tr>
<td>b.</td>
<td>GENERATE Subroutine</td>
<td>97</td>
</tr>
<tr>
<td>c.</td>
<td>INTERRUPT Program</td>
<td>97</td>
</tr>
<tr>
<td>d.</td>
<td>LINK</td>
<td>97</td>
</tr>
<tr>
<td>e.</td>
<td>LOADX Subroutine</td>
<td>101</td>
</tr>
<tr>
<td>f.</td>
<td>MINTOBI Subroutine</td>
<td>101</td>
</tr>
<tr>
<td>g.</td>
<td>OPGEN Subroutine</td>
<td>101</td>
</tr>
<tr>
<td>h.</td>
<td>RIGHT Subroutine</td>
<td>107</td>
</tr>
<tr>
<td>i.</td>
<td>RESTART Program</td>
<td>107</td>
</tr>
<tr>
<td>j.</td>
<td>SAYNUM Subroutine</td>
<td>107</td>
</tr>
<tr>
<td>k.</td>
<td>TBLOOK Subroutine</td>
<td>110</td>
</tr>
<tr>
<td>l.</td>
<td>2BEEP Subroutine</td>
<td>110</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (continued)

VI. CONCLUSIONS AND RECOMMENDATIONS .................................. 114
   A. Cost Analysis ................................................................ 114
   B. Device Improvement Recommendations ...................... 114
   C. Notes on Doing Similar Projects ................................. 116

REFERENCES CITED ................................................................. 118

BIBLIOGRAPHY .................................................................. 120

APPENDIX I  DEVELOPMENT CIRCUITRY ................................. 121
APPENDIX II  DEVICE HARDWARE DOCUMENTATION ............ 128
APPENDIX III  DEVICE SOFTWARE DOCUMENTATION ........ 143
APPENDIX IV  PROCEDURE FOR BURNING EPROMs ............... 196
APPENDIX V  USERS MANUAL .............................................. 200

VITA .............................................................................. 213
LIST OF TABLES

TABLE I  ADDITION PROBLEM SETS................................. 11
TABLE II SUBTRACTION PROBLEM SETS............................ 12
TABLE III MULTIPLICATION AND DIVISION PROBLEM SETS......... 13
TABLE IV 64-WORD SYNTHESIZER VOCABULARY.................... 18
TABLE V DEVICE ADDRESS SUMMARY.............................. 43
TABLE VI DEVICE CONTROL CODES................................ 44
TABLE VII GLOSSARY OF SUBROUTINES............................ 49
TABLE VIII OPERAND RANGE AS DETERMINED BY A-REGISTER..... 105
TABLE IX DEVICE COSTS........................................... 115

APPENDIX II
TABLE I CONNECTOR INFORMATION................................ 140

APPENDIX III
TABLE I SYMBOL TABLE........................................... 144
TABLE II SUBROUTINE HIERARCHY................................. 147
TABLE III PROGRAM LISTING...................................... 157

APPENDIX IV
TABLE I ADDITION PROBLEM SETS................................. 206
TABLE II SUBTRACTION PROBLEM SETS............................ 207
TABLE III MULTIPLICATION AND DIVISION PROBLEM SETS......... 208
LIST OF FIGURES

Figure 1  Math Trainer........................................5
Figure 2  Keyboard Detail....................................6
Figure 3  Device Flowchart (Users view)....................8
Figure 4  Programming Event Sequence....................19
Figure 5  System Overview..................................21
Figure 6  4K RAM Board....................................28
Figure 7  Memory Protect Circuit.............................29
Figure 8  RESET Circuit......................................31
Figure 9  EPROM Circuit.....................................33
Figure 10 Interface Overview.................................34
Figure 11 7-Segment Display Circuit.......................36
Figure 12 Emergency Blanking Circuit.......................38
Figure 13 Operation Sign Matrix.............................39
Figure 14 Operation Sign Circuitry..........................40
Figure 15 Prompt Circuit....................................42
Figure 16 MAIN Subroutine................................42
Figure 17 DRILL Subroutine................................44
Figure 18 PROGRAM Subroutine...............................55
Figure 19 REPORT Subroutine................................59
Figure 20 TEACH Subroutine................................60
Figure 21 BLANK Subroutine................................63
Figure 22 BLANKIT Subroutine...............................64
Figure 23 DISPLAY Subroutine...............................65
Figure 24 GETDIG Subroutine...............................66
Figure 25 GETEM and PACK Subroutines.................................69
Figure 26 SAYDISP Subroutine........................................70
Figure 27 SCAN DISPLAY Subroutine................................72
Figure 28 CHECK SPEECH Subroutine................................73
Figure 29 Circular FIFO (Speech Buffer)...............................75
Figure 30 CHECK TIME & KEY DEBOUNCE Subroutine...........78
Figure 31 Finite State Representation of Key Debounce..........79
Figure 32 SPKEY Subroutine.............................................80
Figure 33 SPKWD and LDPAUS Subroutines...........................82
Figure 34 UNPACK Subroutine...........................................83
Figure 35 WAIT Subroutine................................................84
Figure 36 ADDEM Subroutine.............................................86
Figure 37 BCDADD Subroutine...........................................87
Figure 38 BCDINC Subroutine............................................87
Figure 39 BCDMULT Subroutine..........................................88
Figure 40 BISUB Subroutine...............................................90
Figure 41 COMPARE Subroutine...........................................90
Figure 42 DADD Subroutine...............................................91
Figure 43 GETANS Subroutine............................................92
Figure 44 How HI-LO works.................................................94
Figure 45 HI-LO Subroutine...............................................95
Figure 46 RANDOM Subroutine...........................................96
Figure 47 ERROR Subroutine..............................................98
Figure 48 GENERATE Subroutine........................................99
Figure 49 INTERRUPT Subroutine....................................100
Figure 50 How LOADX Works..............................................102
Figure 51  LOADX Subroutine........................................... 103
Figure 52  MINTOBI Subroutine....................................... 104
Figure 53  OPGEN Subroutine......................................... 106
Figure 54  RIGHT Subroutine.......................................... 108
Figure 55  RESTART Subroutine....................................... 109
Figure 56  SAYNUM Subroutine........................................ 111
Figure 57  TBLOOK Subroutine........................................ 112
Figure 58  2BEEP Subroutine......................................... 113

Appendix I
Figure 1  D2 Board Microcomputer Module........................ 122
Figure 2  4K RAM Board................................................ 126
Figure 3  Memory Protect Circuit.................................... 127

Appendix II
Figure 1  D1 Board with Changes................................. 129
Figure 2  Display Board Circuitry.................................. 131
Figure 3  EPROM Board Circuitry................................. 136
Figure 4  Keyboard Circuitry........................................ 137
Figure 5  Jumper Board................................................ 138
Figure 6  Power Supply............................................... 139

APPENDIX IV
Figure 1  D2 to Prolog Interface................................. 197

APPENDIX V
Figure 1  Math Trainer Layout................................. 201
Figure 2  Keyboard Detail........................................... 202
I. INTRODUCTION

Research has shown that, on the average, unsighted students score lower than do sighted students on standardized achievement tests, such as the SAT.\(^1\) Math skill exhibited a definite deficiency.\(^2\) The area to which this thesis is aimed is that of mathematics and the blind student.

A. The Problem

A blind student is at a disadvantage when compared to the sighted student. The manipulative skills (e.g., scratchwork on paper, problem visualization) that the sighted student routinely employs in working mathematics problems are not as easily employed by the blind student.

Nolan and Ashcroft\(^3\) and, more recently, Brothers\(^4\) reported that blind students average between eight to fifteen percent lower on SAT scores than do sighted students. Brothers\(^5\) attributes this deficit to the arithmetic computational ability of the blind student. Mathematics skills, therefore, is an area where assistance in training the blind student is strongly needed.

B. Project Motivation

The need for assistance in the area of mathematics training of the blind has been demonstrated above. One device that is currently in use to aid visually handicapped students is the Speech-Plus talking calculator. Originally developed by Telesensory Systems, Inc.,\(^6\)
the talking calculator was later modified by The American Printing House for the Blind to include a speech rate control. A survey of schools using talking calculators\textsuperscript{7} reported the following as frequent applications: teaching number concepts; teaching general and remedial mathematics; and use as a replacement for braille calculators, the abacus, or slide rules for those students who can't or won't use these instruments because they are too cumbersome.

The obvious utility of spoken output prompted the author to pursue the creation of a talking mathematics trainer. Although a braille-output trainer is feasible; only about 21\% of visually handicapped students read braille.\textsuperscript{8}

\section*{C. Some Talking Devices}

A review of the appropriate literature revealed no references to a talking mathematics trainer. However, some devices incorporating synthesized speech are worth reviewing.

\subsection*{1. Texas Instruments' "Speak and Spell\textsuperscript{9}"}

The "Speak and Spell", although teaching a different subject, comes remarkably close to what a mass produced talking math trainer would be like. The "Speak and Spell", selling for only $48.00 has a vocabulary of about 200 words. The device verbally presents words to be spelled and the user enters answers via an alphabetic keyboard. The "Speak and Spell also has an alphabetic LED display, used to show the student's entries. The synthesizer used by the "Speak and Spell" is a monolithic device using an advanced linear predictive coding algorithm.
2. The "Speech Plus" Calculator

The "Speech Plus" calculator has been mentioned previously. It is a small calculator designed for the blind student. It contains a Read Only Memory (ROM) circuit that stores the speech. The contents of the ROM are clocked out under control of a ROM Controller. The digital speech signal is routed to a resistor network which serves as a digital to analog converter. A more advanced version of the "Speech Plus" synthesizer was chosen as the synthesizer for the Math Trainer.

3. The Votrax

The Votrax is a phoneme oriented speech synthesizer. With the Votrax, a user can form words by stringing together several basic sounds called phonemes. The use of a phoneme structured device like the Votrax permits an almost unlimited vocabulary. However, each word must be coded by hand, or the phonemes must be chosen by an extremely large program that performs speech synthesis by rule, in order to get a good approximation to human speech.

The aim of the present work is to design and construct a talking mathematics trainer that will assist the visually handicapped student in acquiring basic number skills.
II. FUNCTIONAL SPECIFICATIONS AND MAN-MACHINE INTERFACE

A. Device Overview

A proof-of-concept prototype has been designed to tutor a visually handicapped student in the four basic arithmetic operations: addition, subtraction, multiplication, and division. The device uses speech in order to present the student with an incomplete mathematical sentence such as "three plus two equals blank". The student will use a numeric keypad to enter a choice of what should be in the blank. A sketch of the device is shown in Figs.1 and 2. The device will inform the student of the response's correctness. If the answer is incorrect, the device will repeat the problem. If the student fails to enter the correct response the second time, the device will inform the student of the response's incorrectness and will speak the entire problem with the correct answer in its proper place. If the student's answer is correct, the device will indicate that the response is correct and speak the complete mathematical sentence with the answer in place. After each problem, the device records the correctness of the student's answer and the amount of time that it took the student to respond. After the student has completed all of the problems that the teacher has assigned, the device reports the number of correct answers, the total number of problems attempted, and the total response time.

The teacher is able to select how the problems will be presented to the student. Problem sets may be setup on the basis of: 1) a maximum
FIGURE 1 Math Trainer
CHOICE OF METHOD
OF DEFINING PROBLEM
SETS ENTERED HERE

PACING OPTION
ENTERED HERE

OPERATION
ENTERED HERE

PROGRAMMING
PROMPTS

OPER.

PROB.

PACE

R

T

S

X

P

+

0

0

0

COLUMN PROMPTS

FIGURE 2 - Keyboard Detail. Prompt labels are in both print and braille.
total response time, 2) a Power mode, with a fixed number of problems and unlimited time, or 3) a Speed mode where the student has a fixed amount of time per problem and a fixed number of problems in the problem set.

The teacher is also able to define the contents of the problem sets. In addition to specifying the operation involved (addition, etc.), the teacher can delimit the problem sets using two different modes. In one mode, the teacher supplies the device with the level of difficulty desired. In the other mode, the teacher supplies the upper and lower bounds on the two operands needed to define the problems. The problems are generated randomly within a given problem set. The overall sequence of events in the operation of the device is shown in Figure 3. Each problem set is repeatable, so the teacher can determine which type of problem the student finds the most difficult.

A discussion of the choices made with regard to problem set generation, as well as learning modes, is presented in the following sections.

B. Problem Generation

1. Range mode

In the Range mode, the teacher defines the problem set by keying in the upper and lower limits for each of the independent variables (OPERANDS) of the problem. The device generates a random number between or equal to, the higher and lower limits for the first operand. This random number then becomes the actual value of the first operand. This process is repeated for the second operand. Once the two operands have been determined, the device will perform
FIGURE 3 Device Flowchart (User's View)
the mathematical operations necessary to solve for the dependent variable.

This mode allows the teacher to tailor problem sets to a student's specific needs. If, for example, a student is having trouble solving problems involving multiplication by three, the teacher could have the device present only problems that have three as the second operand. This could be accomplished by setting both of the limits of the second operand equal to three.

2. Level of Difficulty Mode

In the level of difficulty mode, the teacher supplies a level number to the device; that number corresponds to a specific predefined problem set. Those problem sets are arranged so that, for a given operation, they are progressively more difficult as the level number increases.

The Level mode forms a fixed learning program for the student. The teacher needs only to increment the level number, after the student becomes proficient in the present level, in order to proceed sequentially to the next more difficult problem set. Of course, the teacher must train the student in the mathematical mechanics necessary for each problem set.

In selecting the problem set for each level, every effort was made to present the computational concepts in the same order as is currently employed in the elementary schools. One of the major determinants of problem set difficulty is the presence of problems that involve regrouping. Regrouping is a term used in education to denote the occurrence of a carry in addition or a borrow in subtraction.
The device will discriminate between problems that have regrouping for addition and subtraction. (The problem set definitions for addition and subtraction are shown in TABLES I and II.) Because division is normally taught as the conceptual inverse of multiplication, it was decided that the division problem sets would be the inverses of the multiplication problem sets (see TABLE III).

C. Problem Presentation

1. Learning Considerations

   a. Format. The device is designed to be used by students in elementary school; therefore the problem sets will be limited to integer arithmetic. Generally, non-integer (rational) arithmetic is not introduced to the student until later. Since the integers are not closed under division, only problems that have integer quotients are generated.

   It was decided that a signed three digit display would give the device sufficient capability to present the types of problems that are usually taught at the elementary school level.

   Often, problems that are presented in the students' textbooks, have the missing number in positions other than to the right of the equals sign. Because of this, the device will allow the teacher to change the position of the missing number to correspond to any of the three positions in the problem sentence.

   In teaching machines, one often finds that a student is given a couple of attempts at each problem before being scored incorrect. A good example of the above is Texas Instrument's Speak and Spell, which is a spelling trainer. Given that synthesized speech may be
<table>
<thead>
<tr>
<th>Problem Set Level</th>
<th>Range of Operands</th>
<th>Augend</th>
<th>Addend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 to 4</td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td>2</td>
<td>0 to 4</td>
<td>5 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>3</td>
<td>5 to 9</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>4</td>
<td>5 to 9</td>
<td>5 to 9</td>
<td>5 to 9</td>
</tr>
<tr>
<td>5</td>
<td>0 to 9</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>6</td>
<td>0,10,20,...,90</td>
<td>0 to 9</td>
<td>0,10,20,...,90</td>
</tr>
<tr>
<td>7</td>
<td>0 to 9</td>
<td>0,10,20,...,90</td>
<td>0,10,20,...,90</td>
</tr>
<tr>
<td>8</td>
<td>0,10,20,...,90</td>
<td>0,10,20,...,90</td>
<td>0,10,20,...,90</td>
</tr>
<tr>
<td>9*</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>10*</td>
<td>0 to 9</td>
<td>10 to 9</td>
<td>10 to 9</td>
</tr>
<tr>
<td>11</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>12</td>
<td>0 to 99</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>13*</td>
<td>19 to 99</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>14</td>
<td>10 to 99</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>15*</td>
<td>100 to 499</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>16*</td>
<td>0 to 9</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>17</td>
<td>100 to 499</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>18</td>
<td>0 to 9</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>19*</td>
<td>100 to 499</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>20*</td>
<td>10 to 99</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>21</td>
<td>100 to 499</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>22</td>
<td>10 to 99</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>23*</td>
<td>100 to 499</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>24</td>
<td>199 to 499</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>25</td>
<td>100 to 499</td>
<td>-499 to 499</td>
<td>-499 to 499</td>
</tr>
<tr>
<td>26</td>
<td>-499 to 499</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>27</td>
<td>-499 to 499</td>
<td>-499 to 499</td>
<td>-499 to 499</td>
</tr>
</tbody>
</table>

*NOTE: Indicates that problems containing regrouping are not members of the problem set.
### TABLE II

**SUBTRACTION PROBLEM SETS**

<table>
<thead>
<tr>
<th>Problem Set</th>
<th>Range of Operands</th>
<th>Minuend</th>
<th>Subtrahend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1*</td>
<td>0 to 4</td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td>Level 2*</td>
<td>0 to 4</td>
<td>0 to 4</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 3*</td>
<td>5 to 9</td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td>Level 4*</td>
<td>5 to 9</td>
<td>0 to 4</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 5*</td>
<td>0 to 9</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 6**</td>
<td>10 to 99</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>Level 7</td>
<td>0,10,20,...90</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 8</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 9**</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 10**</td>
<td>100 to 499</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 11</td>
<td>100 to 499</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 12**</td>
<td>100 to 499</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>Level 13</td>
<td>100 to 499</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>Level 14**</td>
<td>100 to 499</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>Level 15</td>
<td>0 to 9</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 16</td>
<td>19 to 99</td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td>Level 17</td>
<td>100 to 499</td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td>Level 18</td>
<td>0 to 9</td>
<td>-9 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Level 19</td>
<td>-9 to 9</td>
<td>0 to 9</td>
<td>-9 to 9</td>
</tr>
<tr>
<td>Level 20</td>
<td>-9 to 9</td>
<td>0 to 9</td>
<td>-9 to 9</td>
</tr>
<tr>
<td>Level 21*</td>
<td>-499 to 499</td>
<td>-499 to 499</td>
<td>-499 to 499</td>
</tr>
</tbody>
</table>

*NOTE:* Indicates the given operand ranges are actually those of the augend and addend in the complementary addition problems.

**NOTE:** Indicates that problems containing regrouping are not members of the problem set.
TABLE III
MULTIPLICATION AND DIVISION PROBLEM SETS

<table>
<thead>
<tr>
<th>Problem Sets</th>
<th>Multiplicand</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Range of Operands*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiplicand</td>
<td>Multiplier</td>
</tr>
<tr>
<td>1</td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td>2</td>
<td>0 to 4</td>
<td>0 to 9</td>
</tr>
<tr>
<td>3</td>
<td>0 to 9</td>
<td>0 to 4</td>
</tr>
<tr>
<td>4</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>5</td>
<td>0,10,20,...,90</td>
<td>0 to 9</td>
</tr>
<tr>
<td>6</td>
<td>0 to 9</td>
<td>0,10,20,...,90</td>
</tr>
<tr>
<td>7</td>
<td>0 to 9</td>
<td>10 to 99</td>
</tr>
<tr>
<td>8</td>
<td>10 to 99</td>
<td>0 to 9</td>
</tr>
<tr>
<td>9</td>
<td>10 to 30</td>
<td>10 to 30</td>
</tr>
<tr>
<td>10</td>
<td>0 to 9</td>
<td>-9 to 9</td>
</tr>
<tr>
<td>11</td>
<td>-9 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>12</td>
<td>-30 to 30</td>
<td>-30 to 30</td>
</tr>
</tbody>
</table>

*NOTE: For division the problem is generated by the multiplication of numbers within the ranges in the table. The product is then placed in dividend position.*
difficult to recognize, it was felt that to give the student only one chance per problem would be unduly frustrating. It was decided that the device will give the student two attempts at getting each problem correct.

Without feedback, the student would not be able to learn from making errors. The device provides two forms of feedback. The first is immediate item-by-item feedback. According to Holding, "Separate or item-by-item feedback is very effective in teaching machines":16 After the student enters an answer, the device immediately informs the student of the answer's correctness. The second type of feedback is what Holding terms "accumulated feedback":17 In the device, the feedback consists of reporting the number of correct answers, the number of attempted problems, and the total amount of time the student took to answer the problems in the problem set. Holding points out that this type of feedback is especially good for increasing motivation.

One of the skills that is generally taught, alongside the math skills, is the proper speaking of multidigit numbers. Students are taught to pronounce 214 as "two hundred fourteen", and not as "two one four". The decision was made that the device would pronounce its numbers in the proper way.

The device also has the capability to repeat a problem set verbatim. The major use of this feature would be if the teacher noticed that the student had difficulty with the present problem set. The teacher could then cause the problem set to be repeated, and observe the types of problems that give the student the most difficulty.
b. Pacing Options. The teacher may choose three options that govern how the device limits problem sessions. In the first option (the Timed option), the teacher limits the time the student works on a problem set. When the time runs out, the problem session ends. In the second option (Speed), the teacher is able to limit the amount of time per problem attempt, as well as the total number of problems. If the time per attempt runs out, the device acts as if the student had entered an incorrect answer; if the maximum number of problems has been reached the problem session ends. In the third option (Power) the limiting factor is the number of problems. The teacher limits the number of problems in the session, but the student may take any amount of time in responding to each problem. When the maximum number of problems has been completed the problem session ends.

2. Man-Machine Considerations

The foremost feature of this device is that it has a voice output. The Speech Plus talking calculator has been on the market for a few years. The speech synthesizer that is used in the present device is a more advanced version of the one used in that calculator, and has a 64 word fixed vocabulary, shown in TABLE IV.

Using the vocabulary of the synthesizer several standard phrases were created. When a problem has been answered correctly, a string of three low tones (word number 61\textsubscript{10}) are said by the synthesizer. An incorrect response elicits two high tones (word number 62\textsubscript{10}). These arbitrary tone sequences were chosen because no semantically relative true-word combinations could be found. The words "go please", however, served nicely as a general purpose audio prompt.
If a sighted student forgets the current problem, that student can simply look at the problem again. A blind student listening to a verbal problem should have the same opportunity. For this reason a repeat problem presentation button has been incorporated into the device.

Because many students are familiar with the Speech Plus calculator, the device's keyboard is designed with the numeric keys in the same configuration as the calculator's. The only difference being the presence of an enter key and a minus key (for negative number entry).

It was also decided to incorporate auditory feedback of keyboard entry into the device. As each key is depressed, the device speaks the key's designation. Keys which do not have descriptions in the vocabulary will, when depressed, cause a tone to be sounded.

In addition to the keyboard echo feature, the device also has a keyboard training drill. This drill will ask the student to depress the keys on the keyboard in a certain order. If the student hits the right key, that key will echo its designation and sound a tone. Wrong keys will only echo their designations. It was decided that the device's display should be as large as possible, so that a low vision student might benefit from the visual input. The main reason for having the display, however, is to aid a sighted teacher in watching the students progress when the student is using headphones.

D. Teacher Programming

Before each problem session, the teacher will have to set up (program) the device in the desired configuration. Parameters such
as pacing mode, operation to be performed, location of missing number, number of problems, level of difficulty, etc. must be supplied to the device. Verbal asking of the required questions is not possible with the limited vocabulary of the device (see TABLE IV). Therefore, a scheme using a prompt beside a written statement (in both braille and print) of the parameter to be entered was decided upon. The prompt is an illuminated LED next to a print and brailer label. Later, if funds are available, solenoids will replace the LEDs to allow a non-sighted person to program the device. For details of teacher programming see the users manual in Appendix V. It was felt that, to eliminate student confusion, the programming controls should be located away from the controls the student will use (see Fig. 2). Fig. 4 illustrates the sequence of events involved in setting up the device by the teacher.
### TABLE IV

**64-WORD SYNTHESIZER VOCABULARY**

<table>
<thead>
<tr>
<th>Data Code (decimal)</th>
<th>Word</th>
<th>Data Code (decimal)</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>zero</td>
<td>032</td>
<td>times</td>
</tr>
<tr>
<td>001</td>
<td>one</td>
<td>033</td>
<td>over</td>
</tr>
<tr>
<td>002</td>
<td>two</td>
<td>034</td>
<td>equals</td>
</tr>
<tr>
<td>003</td>
<td>three</td>
<td>035</td>
<td>point</td>
</tr>
<tr>
<td>004</td>
<td>four</td>
<td>036</td>
<td>overflow</td>
</tr>
<tr>
<td>005</td>
<td>five</td>
<td>037</td>
<td>clear</td>
</tr>
<tr>
<td>006</td>
<td>six</td>
<td>038</td>
<td>percent</td>
</tr>
<tr>
<td>007</td>
<td>seven</td>
<td>039</td>
<td>and</td>
</tr>
<tr>
<td>008</td>
<td>eight</td>
<td>040</td>
<td>seconds</td>
</tr>
<tr>
<td>009</td>
<td>nine</td>
<td>041</td>
<td>degrees</td>
</tr>
<tr>
<td>010</td>
<td>ten</td>
<td>042</td>
<td>dollars</td>
</tr>
<tr>
<td>011</td>
<td>eleven</td>
<td>043</td>
<td>cents</td>
</tr>
<tr>
<td>012</td>
<td>twelve</td>
<td>044</td>
<td>pounds</td>
</tr>
<tr>
<td>013</td>
<td>thirteen</td>
<td>045</td>
<td>ounces</td>
</tr>
<tr>
<td>014</td>
<td>fourteen</td>
<td>046</td>
<td>total</td>
</tr>
<tr>
<td>015</td>
<td>fifteen</td>
<td>047</td>
<td>please</td>
</tr>
<tr>
<td>016</td>
<td>sixteen</td>
<td>048</td>
<td>feet</td>
</tr>
<tr>
<td>017</td>
<td>seventeen</td>
<td>049</td>
<td>meters</td>
</tr>
<tr>
<td>018</td>
<td>eighteen</td>
<td>050</td>
<td>centimeters</td>
</tr>
<tr>
<td>019</td>
<td>nineteen</td>
<td>051</td>
<td>volts</td>
</tr>
<tr>
<td>020</td>
<td>twenty</td>
<td>052</td>
<td>ohms</td>
</tr>
<tr>
<td>021</td>
<td>thirty</td>
<td>053</td>
<td>amps</td>
</tr>
<tr>
<td>022</td>
<td>forty</td>
<td>054</td>
<td>hertz</td>
</tr>
<tr>
<td>023</td>
<td>fifty</td>
<td>055</td>
<td>DC</td>
</tr>
<tr>
<td>024</td>
<td>sixty</td>
<td>056</td>
<td>AC</td>
</tr>
<tr>
<td>025</td>
<td>seventy</td>
<td>057</td>
<td>down</td>
</tr>
<tr>
<td>026</td>
<td>eighty</td>
<td>058</td>
<td>up</td>
</tr>
<tr>
<td>027</td>
<td>ninety</td>
<td>059</td>
<td>go</td>
</tr>
<tr>
<td>028</td>
<td>hundred</td>
<td>060</td>
<td>stop</td>
</tr>
<tr>
<td>029</td>
<td>thousand</td>
<td>061</td>
<td>tone(low)</td>
</tr>
<tr>
<td>030</td>
<td>plus</td>
<td>062</td>
<td>tone(high)</td>
</tr>
<tr>
<td>031</td>
<td>minus</td>
<td>063</td>
<td>oh</td>
</tr>
</tbody>
</table>
1) Teacher turns device on.
2) Teacher chooses either Range or Level mode.
3) The Pacing Option (Timed, Speed, Power) is selected.
4) If the Time Option was selected the teacher enters the total time limit (via the keypad) in minutes and seconds (up to 99 minutes and 99 seconds).
5) If the Speed Option was selected the teacher enters in the time limit per attempt in minutes and seconds (up to 20 minutes).
6) If the Timed Option has not been selected the teacher enters the number of problems in the problem session via the keypad.
7) The teacher keys in the operation (+, -, x, or ÷).
8) If the Level Mode was selected the teacher keys in the level of difficulty number.
9) If the Range Mode was chosen the teacher enters the high and low limits of the first operand followed by the high and low limits of the second operand.

FIGURE 4 - Programming Event Sequence
III. SYSTEM DESIGN

The math trainer, described in Chapter II, consists of a speech synthesizer for voice output, a keyboard for data entry, a visual display for problem presentation, and prompts to aid the teacher in programming problem set information into the device. All of these components are tied together through a controller to form a complete system. The system is designed so that the controller can coordinate the actions of each of the components.

A. System Requirements

The system consists of five basic components; the controller, the keyboard, the LED display, the speech synthesizer, and the prompts. These subsystems are functionally tied together as shown in Fig. 5.

Specifications for the controller required that it must be able to control the activities of the other components. It must be able to store and manipulate the data received from the keyboard and provide data for the output subsystems (display, synthesizer, and prompts). A microcomputer was the most logical choice for the controller.

The second major component is the keyboard. There are 22 keys on the keyboard; ten digits, minus, enter, repeat, and nine keys dedicated to programming the problem sets. (See Fig.2 in Chapter II). The controller must be able to determine which, if any, of these keys is depressed. The keys themselves must be debounced (i.e., so that a single key closure is not interpreted as multiple closures).
FIGURE 5 - System Overview
The third major subsystem is the LED display, used to present the problem visually. The display must have three numbers, each consisting of a sign and three digits. In addition, it must have an operation sign (+, -, x, +) and an equals sign. The display should be as large as possible, as some users will have some useful vision. Moreover, having a large display will allow the teacher to monitor a student's progress from a distance.

The speech synthesizer subsystem is what makes the device unique. The synthesizer must present the problems to the student, echo the keys as they are depressed, provide audio feedback, and report the student's score. The synthesizer must provide intelligible speech at a rate that the user can comprehend.

The last subsystem is the prompt system. This system is designed to say the questions that the synthesizer cannot say, due to its limited vocabulary. If solenoids are eventually used for the prompts, they must be small enough so that if they are placed side-by-side they will not take up too much room in the device. The solenoids must also be rated for continuous duty, low power, and a convenient voltage (five volts would be ideal). At present, small, inexpensive solenoids are not available, so discrete LEDs were substituted. The LEDs must have sufficient brightness to be seen in a normal classroom environment. Use of LEDs instead of solenoids, would not help an unsighted teacher to program the device, but would still allow for proof-of-concept evaluation of the device by blind students.

In addition to the five components described above, the device
must have a DC power supply that is capable of meeting the current and voltage requirements of the subsystems. Also, the device should be portable; therefore, the physical size and weight of the device should be low enough for easy transportation. Although the device is intended as a proof-of-concept prototype, it should, nonetheless, be made as economical as possible.

B. Subsystem Selection

After considering the microcomputer systems that were available, the Motorola 6800 was chosen to act as the controller. A large amount of software is available for the 6800. Also, the 6800 is suited for data manipulation of the type required by the device. For example, it has instructions that perform Binary Coded Decimal (BCD) arithmetic; BCD arithmetic is often used in calculators and devices with large amounts of numeric input and output. Perhaps the most important reason for using the 6800 is its availability. The Perceptual Alternatives Laboratory supplied the author with a Motorola 6800 based D2 evaluation system. That microcomputer is a program development system that allows the programmer to enter, debug, and single step through programs. After development was in progress, the Electrical Engineering Department offered the use of a 6800 based D1 system. It was decided to continue to use the D2 for program development, and to incorporate the simpler D1 into the device as the controller.

The keyboard requires 22 switches; to cut down on the number of switch inputs to the controller, a multiplexing scheme was used.
The switches, themselves, had to be made such that they could be touched without being activated. This feature allows a user to tactually localize the position of the various keys without inadvertently entering incorrect information. Keys that were obtained from a used keyboard satisfy the above condition. These keys were used in the Math Trainer.

For the display, Litronix DL 747\textsuperscript{2} seven-segment displays was chosen. These displays (which are common anode) are easily multiplexed. The decision to multiplex the display reduces the number of components required. In addition to the seven-segment displays, an operation display is required. It was decided to make that display from discrete LEDs, since nothing could be found that was suitable for the task.

The next major subsystem is the speech synthesizer. The decision was made to use the Telesensory Systems Incorporated model S2B 64-word voice synthesizer\textsuperscript{21} (see TABLE IV). This synthesizer contains a vocabulary rich enough to be able to say any number up to 999,999 properly. It can also say the words required to present problems. The prompts substitute for most of the additional needed words that are not in the speech board's vocabulary. Words can be added to its vocabulary; but, TSI charges about $200 to add words available on other models, and considerably more than $200 for totally new words.\textsuperscript{22} It was felt that the standard model would be sufficient for a proof-of-concept device. The TSI synthesizer requires 6 word-select lines and two handshake lines. This last subsystem, together with the previously mentioned ones, use up the control lines available on the D1 and D2 computer boards.
The last subsystem is the prompt system. Funds for solenoids were not available, so, LEDs were used instead.

To power the device a Godbout multivoltage power supply was selected. Rated to supply four amps at five volts and 0.5 amps at +12 volts and -12 volts, this supply was chosen for its low cost and availability.
The development of the hardware for the Math Trainer was done in a stepwise fashion. All of the software development was done using a Motorola MEK 6800 D2 kit\textsuperscript{23}, driving a prototype interface circuit. The prototype interface circuit was constructed on breadboarding sockets. After the majority of the software was completed, the interface circuit was rebuilt in its final form using wire-wrap sockets. Finally, after all of the control programs were finished the software was transferred to EPROMs and a MEK 6800 D1\textsuperscript{24} board was modified to serve as the controller.

A. The Development System

1. The Controller

The MEK 6800 D2 board (schematic shown in Fig. 1 of Appendix I) was used as the device controller for the Math Trainer during software development. It is well suited to program development. The D2 kit can have as many as 512 bytes of user RAM on board, with provisions for up to 16K bytes of off-board memory. Programs can be entered and changed via a keypad, and an LED display integral to the D2 kit permits verifying contents. Programs may be stored using an audio type recorder. The D2 kit has Peripheral Interface Adapter (PIA) that was used to interface to the Math Trainer's hardware.

2. Memory Expansion

The 512 bytes of on-board RAM available on the D2 kit was quickly found to be insufficient to contain the expanding software.
An expansion memory board, designed by Brian Garvey, was added to the D2 board. (See Fig. 6). This memory board can accommodate up to 8K bytes (in 1K increments), of which only 4K bytes were actually installed. The author also designed and installed a memory protect (write disable) circuit to prevent faulty programs from self-destructing. The protect circuit is shown in Fig. 7. A full schematic of the 4K RAM board is shown in Fig. 2 of Appendix I.

B. Final System Design

1. The Controller

The D2 kit used for software development was deemed unsuitable for use as the final device controller. Much of D2 would not be needed in the final version (e.g., the cassette interface, the Hex keypad, etc.). A MEK 6800 D1 kit was supplied for the author's use by the Electrical Engineering Department of the University of Louisville and was used as the controller. A schematic of the D1, incorporating the needed changes is shown in Fig. 1 of Appendix II.

Unlike the D2, the D1 is a single board computer. It can hold up to 512 bytes of user RAM (although only 256 bytes are required for the controller). Because the user PIA addresses differ for the D1 and D2, the software was changed to reflect that fact before the EPROMs were programmed. The only major changes required to the D1 board involved the RAM select and \texttt{RESET} circuits. The D1's address decoding for the user's RAM is incomplete. When certain parts of the control program are addressed by the microprocessor, user RAM is also selected. Control program range from \$2000 to \$2FFF while user RAM extends from \$0000 through \$00FF. A RAM ENABLE signals was
*NOTE: ALL WE tied to R/W on D2.

FIGURE 6 - 4K RAM Board
FROM 6800
R/W

PROTECT
IN4148

1kΩ

+5v

TO G1 INPUT OF 74LS138

Ø2

IN1118

FIGURE 7 - Memory Protect Circuit
generated by OR'ing together $A_{15}, A_{14}, A_{13}$. This signal was generated on the EPROM board and brought onto the D1 through pin $E$. Normally, $E$ contains BUS AVAILABLE (BA) but BA was cut where it leaves the microprocessor. $A_{15}$ was cut before reaching the user RAM's. The $A_{15}$ input to two RAM chips was replaced with RAM ENABLE so the RAMs would only be selected if the three most significant address lines were low. The other major change to the D1 involved cutting the output of the on-board reset circuit so that an external RESET signal could be brought in from the edge connector. This circuit along with the other Math Trainer circuits can be found in Appendix II.

A power-on reset circuit was built for the controller utilizing a design shown in the Motorola 6800 Applications Manual. The actual circuit used is shown in Fig. 8. The user can also reset the device by pushing the normally closed momentary switch (RESTART). That switch interrupts the power to the LM555 causing a power-on sequence.

2. Firmware

A working device cannot store its control programs in RAM, as they would be lost every time power were removed. Control programs must be stored in read only memory (ROM). For the Math Trainer, 2708 erasable programmable ROMs (EPROMs) were used. These EPROMs can be erased and reprogrammed many times.

When the control programs were written, they were located in memory from $2000$ to $2FFF$. The 6800 needs some memory at $FFFF$ to $FFFF$ for its Reset and Interrupt vectors. It was decided to use incomplete address decoding for the EPROMs, to allow the Restart and Interrupt vectors to be located in the EPROMs. The circuit used
FIGURE 8 - Reset Circuit
for this is shown in Fig. 9. When the 6800 addresses $FFFE and $FFFF to retrieve the Reset vector, the EPROM at $2FFE and $2FFF supplies it. In a like way, the Interrupt vector is retrieved.

The EPROMs were programmed under the control of a D2 computer using a PROLOG EPROM Programmer. After loading the data to be programmed into the D2's RAM, a program, written by the author and Mr. Jimmy Ryan, was executed. The program sent the data over a RS232 link at 1200 baud to the PROLOG. A diagram showing the circuit used for linking the PROLOG and the D2, and the program appear in Appendix IV.

3. Interface Circuitry

a. Overview. The interface circuitry described in this section includes the LED display, the prompts, the keyboard, the speech synthesizer controls, and the circuitry required to connect those devices to the microprocessor controller. The design used for the Math Trainer needs only one PIA and is shown in Fig.10. The PIA is divided into two parts, with one part controlling the speech synthesizer and the other part controlling the remainder of the interface. The connections to the synthesizer are straightforward. The busy line is sensed by CA1 and the completion of a word causes an interrupt. The synthesizer is triggered by CA2 (the start line).

The remaining interface is somewhat involved. A four bit device address is sent to a four-to-sixteen-line decoder. The four bit address selects one output device (e.g., 7-segment display, operation sign latch, Prompt latch, etc.) and two input switches. The output data for the displays are contained on the four-bit data
FIGURE 9 - EPROM Circuit
FIGURE 10 - Interface Overview
bus. The positions of the selected pair of switches are sensed by PB0 and PB1. The microprocessor can sequentially output the device addresses and the data to those devices while simultaneously bringing in switch information.

C. Display and Keyboard Circuitry

1. Seven Segment Displays

The Math Trainer requires an LED display for visual problem presentation. Litronix DL 747 displays were chosen for numeric displays. Multiplexed by the processor, the DL 747s were configured as shown in Fig.11. The 27 ohm resistors were chosen to allow the 7446 to sink as much current as safely possible (40 ma per segment). The low duty cycle (1/16) prevents the average current through the display form exceeding the 30 ma/segment limit.

2. Minus and Equal Signs

The display also contains three minus signs and one equals sign. These signs are constructed using DL 747s in the circuit shown in Fig.11 with the exception that some of the segments are left unconnected. To make a minus sign only segment g (the center bar) is connected. For an equals sign segments a and g are connected into the circuit and the DL 747 is lowered (with respect to the other 7-segment displays) to line up the equals sign.

3. Emergency Blanking Circuit

The average current through the 7-segment displays is within limits as long as the display is multiplexed by the 6800. If the microprocessor halts for some reason, one of the LED displays would be on continuously, thus destroying it. To prevent this a circuit using
FIGURE 11-7-Segment Display Circuit
half of a LM 556 timer was built (see Fig. 12). In this circuit, the timer is set up as a one-shot multivibrator. The 556 triggers when the last device address is selected. The 556's output then goes high, powering the segment driver (7446). The time constant is adjusted so that the 556 times out after 1.5 scan cycles (at 50 cycles/sec). If the display is being multiplexed, the timer is continuously reset and retriggered. If multiplexing stops, the 556 times out and turns off the display driver.

4. Operation Sign

The need for an operation sign (displaying +, -, x, ÷) presented a problem. No commercially available display was both inexpensive and large enough. The author constructed an operation display using discrete LEOs. The LEOs were arranged in a format shown in Fig. 13. The discrete LEOs were divided into five groups. To simplify connections, the LEOs within a group (and their current limiting resistors) were tied together as a unit. The way the LEOs were grouped, if one member were on, the remainder of the group should be on (even if they had been wired individually) to display the operation sign properly. The data to the operation sign was latched, rather than multiplexed; the driver circuitry is shown in Fig. 14. Only one LED is in group 5 (the center of the matrix), and that LED is wired on continuously, and also serves as a DC power indicator.

5. User Prompts

The teacher has to answer a string of questions to program the Math Trainer. To aid in knowing which question is being asked, a system of LED prompts is used. When an input is required, an LED lights
FIGURE 12 - Emergency Blanking Circuit

\[ R = 270 \, \text{k}\Omega \]

\[ C = 0.1 \, \text{\mu F} \]
ADDITION = GROUPS 2, 3, 4, 5
SUBTRACTION = 3, 5
MULTIPLICATION = 1, 5
DIVISION = 3, 4, 5

FIGURE 13 - Operation Sign Matrix
FIGURE 14 - Operation Sign Circuitry

NOTE: (") denotes latched data.
up next to a label explaining what information is needed. If the information is to be input to keys other than the keypad keys, an LED lights up below the column of keys to be used. Thus, two sets of LEDs are used: one for the labels and one for the key columns. Within a set, the LEDs are mutually exclusive of one another. Like the operation sign, the prompts are latched and not multiplexed. The prompt latch is located at device address 14_{10}. The prompt circuit is shown in Fig.15.

6. Interface Summary

The LED display is designed to present arithmetic problems to the user. To do this, it displays three numbers (each consisting of three digits and a minus sign), an operation sign, and an equals sign. The prompt latch, which drives the prompt LEDs to aid device programming by the teacher is also treated as part of the display. (Device address information for the displays appears in TABLE V). The Seven-Segment displays take a BCD code input. In addition, all of the displays (including prompts) interpret an all ones (15_{10}) input as a blanking code. The required driving codes of the other display devices appears in TABLE VI.
4 Bit Data Bus

4 to 16 Decoder

Prompt Number

Label Prompts

Column Prompts

FIGURE 15 - Prompt Circuit
**TABLE V**  
DEVICE ADDRESS INFORMATION

<table>
<thead>
<tr>
<th>Element:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Display</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>

**Keyboard:**

<table>
<thead>
<tr>
<th>KEY #</th>
<th>KEY NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>minus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Address</th>
<th>Key #</th>
<th>Display</th>
<th>Dis. Buf. Ad.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>A</td>
<td>$006</td>
</tr>
<tr>
<td>1</td>
<td>2.3</td>
<td>B</td>
<td>$007</td>
</tr>
<tr>
<td>2</td>
<td>4.5</td>
<td>C</td>
<td>$0008</td>
</tr>
<tr>
<td>3</td>
<td>6.7</td>
<td>D</td>
<td>$0009</td>
</tr>
<tr>
<td>4</td>
<td>8.9</td>
<td>F</td>
<td>$000A</td>
</tr>
<tr>
<td>5</td>
<td>A,B</td>
<td>G</td>
<td>$0008</td>
</tr>
<tr>
<td>6</td>
<td>C,D</td>
<td>H</td>
<td>$000C</td>
</tr>
<tr>
<td>7</td>
<td>E,F</td>
<td>*</td>
<td>$000D</td>
</tr>
</tbody>
</table>

**KEY # | KEY NAME**

<table>
<thead>
<tr>
<th>KEY #</th>
<th>KEY NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Range Mode</td>
</tr>
<tr>
<td>C</td>
<td>Level Mode</td>
</tr>
<tr>
<td>D</td>
<td>Timed Option</td>
</tr>
<tr>
<td>E</td>
<td>Speed Option</td>
</tr>
<tr>
<td>F</td>
<td>Power Option</td>
</tr>
<tr>
<td>10</td>
<td>+</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>+</td>
</tr>
<tr>
<td>16</td>
<td>ENTER</td>
</tr>
<tr>
<td>17</td>
<td>REPEAT</td>
</tr>
</tbody>
</table>
### TABLE VI

**DISPLAY DEVICE DRIVING CODES**

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Code</th>
<th>Resulting Display</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8421</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minus &amp; equals sign</td>
<td>1000</td>
<td>On</td>
</tr>
<tr>
<td></td>
<td>1111</td>
<td>Off</td>
</tr>
<tr>
<td>Operations Display</td>
<td>0111</td>
<td>Plus</td>
</tr>
<tr>
<td></td>
<td>1011</td>
<td>Minus</td>
</tr>
<tr>
<td></td>
<td>1101</td>
<td>Times</td>
</tr>
<tr>
<td></td>
<td>1110</td>
<td>Divide</td>
</tr>
<tr>
<td></td>
<td>1111</td>
<td>Off</td>
</tr>
<tr>
<td>Prompts</td>
<td>0000</td>
<td>Keypad Prompt On</td>
</tr>
<tr>
<td></td>
<td>0001</td>
<td>Prob. Set Prompt On</td>
</tr>
<tr>
<td></td>
<td>0010</td>
<td>Pacing Option Prompt On</td>
</tr>
<tr>
<td></td>
<td>0011</td>
<td>Min:Sec. Prompt On</td>
</tr>
<tr>
<td></td>
<td>0100</td>
<td>No. of Prob. Prompt On</td>
</tr>
<tr>
<td></td>
<td>0101</td>
<td>Operation Prompt On</td>
</tr>
<tr>
<td></td>
<td>0110</td>
<td>Ans. Pos. Prompt On</td>
</tr>
<tr>
<td></td>
<td>0111</td>
<td>Level Prompt On</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>Hi. Lim. Prompt On</td>
</tr>
<tr>
<td></td>
<td>1001</td>
<td>Lo. Lim. Prompt On</td>
</tr>
<tr>
<td></td>
<td>1111</td>
<td>All Prompts Off</td>
</tr>
</tbody>
</table>
V. SOFTWARE

Currently, the popularity of modular programming is on the rise. Dividing a large program into smaller sections does have advantages. A small section of a program with a single function is easier for a programmer to work with because it is possible to simultaneously grasp all of the interrelationships present. Also, as each module is completed, it can be tested before it becomes part of the larger program.

This modular approach was employed in developing the software for the mathematics trainer. The control program is arranged in a hierarchical structure. At the top of that structure is a program called MAIN. MAIN is a module (or subroutine) that calls upon other modules which perform various functions. These subroutines may also call other subroutines. Generally speaking, anytime a large function was required in a program, the program for that function was broken down into smaller subroutines.

A. Software Development

The only software development tool available was the J-BUG monitor program on the D2 microcomputer. The lack of an assembler within JBUG, meant that all programs had to be assembled by hand. Consequently, it was desirable that program changes be performed without having to hand reassemble the code. As a result of this, program corrections were made by using patches. A patch is branch from the normal path, to the program correction, followed by a branch
back to the original path. As programs develop, their sizes change. With limited memory, these size changes require that subroutine locations be shifted to efficiently utilize the limited memory space. Any shifting of subroutine locations would normally require that the program be reassembled; however, the use of relative branching and Link (a programming technique) prevents this.

Any discussion of the device's software must be prefaced by a description of the internal architecture of the 6800 microprocessor. The 6800 has three internal data storage registers; a 16-bit index register (x-reg.), and two eight-bit accumulators (the A and B registers). The index register is often used as a pointer to address locations in memory. In addition there are three registers whose presence is somewhat invisible to the programmer: the condition code register (contains the carry bit, overflow bit, etc.), the stack pointer (keeps track of the information in the stack, a LIFO buffer), and the program counter (points to the location of the next instruction).

B. The Software

The software can be divided into four major functional divisions: control routines, input/output routines, arithmetic routines, and special function routines.

Control subroutines act as high level administrators of the device's operations. As various functions need to be performed, the control routines call the needed subroutines. For example, in the teaching mode, TEACH is the control program; it would call GENERATE (to set up the next problem), GETANS (to solve for the answer), and SAYDISP (to say the problem). For a glossary of the control routines
and the other routines see TABLE VII.

The next major functional subdivision involves the transferring of information to and from the various input/output devices. Because the trainer requires simultaneous display output and keyboard input, a single routine had to be able to perform both input and output functions; that routine is SCAN. The information to be output by SCAN is stored in two buffers, the speech buffer (SPKBUF) and the display buffer (DISBUF). The other I/O routines either load information into the buffers (e.g., DISPLAY and SPKWD) or call SCAN repeatedly to bring in a series of keystrokes (e.g., GETDIG).

The third major division of the software is the arithmetic routine division. The 6800 microprocessor is limited in the types of arithmetic functions it can perform. To overcome these limitations and to eliminate repetition of instruction sequences, various arithmetic subroutines were created. To supplement the 6800's instruction set, double-byte binary addition and subtraction subroutines (DADD and BISUB) were added to the programming repertoire.

It was decided that BCD arithmetic should be used in generating the problems. The motivation to use BCD representation stemmed from a need to do manipulations upon various digits of the number (when checking for regrouping, for example) that would be difficult to perform if the numbers were in a binary format. The BCD format employed is a 2-byte representation, with the sign bit occupying the most significant bit of the high order byte. The hundreds digit is contained in the low nibble of the high order byte. The low order byte contains the ten's and one's digits in the high and low nibbles, respectively.
Various subroutines were written in order to perform BCD arithmetic. BCDADD does simple unsigned additions, and ADDEM uses BCDADD to do signed BCD addition. Signed BCD multiplications are performed using BCDMULT. In solving for the answers to the problems created by GENERATE, GETANS calls various BCD subroutines. Magnitude comparisons of BCD numbers can be done by using HI-LO and COMPARE. To generate random numbers, a subroutine (RANDOM) was developed using an 11 stage linear-feedback shift register, realized in software.

The last major software division covers the subroutines that are labeled as special function subroutines. Those routines result from modularizing the control routines. The subroutines RIGHT and ERROR are called by TEACH to signal correct and incorrect student answers. They could have been built into TEACH, but having them separate allowed TEACH to be a simpler module. In a similar way, RESTART is the initializing module for MAIN. Anytime a reset occurs, RESTART is called and, in turn, calls MAIN.

A glossary of subroutines is shown in TABLE VII, and the calling hierarchy of those routines is shown in TABLE I of Appendix III. A functional description of the subroutines follows, with first level subroutine calls identified. The commented disassembled listing of the entire program is shown in Appendix III.

1. Control Subroutines

These subroutines form the control structure of the math trainer's software.

a. MAIN Program. MAIN orchestrates the sequencing of the device's functions (see Fig. 16). The order of the functions
**TABLE VII**

GLOSSARY OF SUBROUTINES, ARRANGED BY FUNCTIONAL DIVISION

**Control Subroutines**

- **DRILL** - trains the student to use the keyboard.
- **MAIN** - sequences the calling of the other control routines.
- **PROGRAM** - allows the teacher to define the problem set via the keyboard.
- **REPORT** - gives information on the student's performance on the current problem session.
- **TEACH** - presents problems to the student and keeps track of the student's performance.
**Input/Output Subroutines**

BLANK - erases the contents of the LED display buffer.

BLANKIT - erases the missing number from the LED display buffer.

DISPLAY - puts the present problem, including the answer, into the LED display.

GETDIG - retrieves up to 6 digits from the keyboard while outputting the contents of the speech and display buffers.

GETEM - prompts for input and calls GETDIG to retrieve digits.

LDPAUS - loads a variable length pause (up to five seconds) into the speech buffer.

PACK - takes three digits and the sum entered via the keyboard and compresses them into a two-byte signed BCD number.

SAYDISP - takes a two-byte number and loads the words needed to say it into the speech buffer.

SCAN - outputs the contents of the display and speech buffers until a single key is pressed or a pre-specified amount of time has passed. This is the routine that actually performs all of the I/O operations.

SPKEY - loads the name of the pressed key into the speech buffer (used for echoing entries).

SPKWD - loads the word in the A-register into the speech buffer.

UNPACK - takes a signed two-byte BCD number and expands it and stores it in the display buffer.

WAIT - holds up program execution while outputting display and speech buffer contents. Execution is delayed until WAIT empties the speech buffer.
Arithmetic Subroutines

ADDEM - performs two-byte signed BCD addition.

BCDADD - performs two-byte unsigned BCD addition.

BCDINC - (BCD) increments the contents of two memory locations.

BCDMULT - performs signed BCD multiplication.

BISUB - performs two-byte binary subtraction.

COMPARE - performs two-byte signed BCD comparison.

DADD - performs two-byte binary addition.

GETANS - solves for answer to problem.

HI-LO - determines if random number is between two limits.

RANDOM - generates a signed three digit random decimal number.

Special Function Subroutines

ERROR - causes the display to blink and an error tone sequence to be sounded.

GENERATE - randomly picks the numbers for a problem within limits provided by the teacher.

INTERRUPT - is called by an interrupt when the speech synthesizer has finished saying a word.

LINK - allows easy program relocation.

LOADX - allows parameter passing.

MINTOBI - converts data entered as minutes and seconds to binary seconds.

OPGEN - randomly picks a single number (when in the level of difficulty mode) using the definition supplied by TBLOOK (see below).

RIGHT - causes a tone sequence indicating correct to be sounded.

RESTART - initializes the math trainer on power up or when the reset button is pressed.

SAYNUM - loads the correct pronunciation of a three digit number into the speech buffer.

TBLOOK - looks up the problem set definition from the level of difficulty supplied by the teacher.

2BEEP - loads the words "GO PLEASE" into the speech buffer.
Call TBLOOK for problem set defin.

Seeds are generated by PROGRAM.

*NOTE: To get back to PROGRAM user must hit RESET.

FIGURE 16 - MAIN Subroutine
called by MAIN are: PROGRAM (allows the teacher to define the problem set), DRILL (the user's keyboard training drill), TEACH (the subroutine that presents the problems), and REPORT (the subroutine that reports on the student's performance). In addition, MAIN insures that the seeds used for the random number generator (used to randomly generate problems) are not equal. The problem set definitions are looked up by TBLOOK when the machine is in the Level of Difficulty mode. An entire problem set can be repeated by pushing the repeat button after the report has been given.

b. DRILL Subroutine. DRILL is a keyboard familiarization drill for the student. The device says N-"please" via SPKEY, where N is the name of a key (see Fig. 17). The student then attempts to push the named key. If the student doesn't push the named key, the pushed key's name is echoed and, after a pause, the key is again requested. If the student does press the correct key, the key name is echoed and a tone is sounded. DRILL then proceeds to the next key, and sequentially steps through the keys, zero to nine, and then the minus key.

c. PROGRAM SUBROUTINE. The PROGRAM subroutine shown in Fig. 18 prompts the teacher to enter the information that determines which problems are to be presented and how those problems are to be presented. In addition, PROGRAM also generates the random number seeds which are used for generating random problems by counting the number of scans that take place before the teacher responds. Information entered under control of PROGRAM, is then used by the TEACH subroutine, which presents the selected problems to the student.
FIGURE 17 - DRILL Subroutine
FIGURE 18 - Program Subroutine
Add Timlim to self 50 times to get number of SCANS(DADD)

TIMLIM over flow

N

Option=Speed?

Y

Timlim=$FFFF

#PROBS

N

Option=Timed?

Y

Prompt num. of problems, say "Go please" (2BEEP)

Set Digits from Keyboard (GETDIG)

PACK Digits into MAXPROBS

PROGRAM 3

FIGURE 18 - Continued
PROGRAM 3

Prom. oper. sign say "Go please"(2BEEP)

SCAN Keyboard

Echo Keyboard(SPKEY)

N $10 < KEY < $13?

OPSIGN=KEY

Prompt missing num. say "Go please"(2BEEP)

GetDigits from Keyboard(GETDIG)

PACK Digits into MISSING#

MODE=Level?

Prom. Level of diff. say "Go please"(2BEEP)

Get digits from keyboard(GETDIG)

PACK digits into LEVEL

Prompt&Load HILIM1
Prompt&Load LOLIM1
Prompt&Load HILIM2
Prompt&Load LOLIM2
(using GETEM,PACK)

RTS

FIGURE 18 - Continued
d. REPORT Subroutine. The REPORT subroutine reports how well the student performed on the last problem set. The total number of correct problems, the number of problems attempted, and the total amount of time spent by the student are reported both visually (in the LED display) and vocally (via the synthesizer). The flowchart of REPORT appears in Fig. 19.

e. TEACH subroutine. The TEACH subroutine quizzes the student on arithmetic problems. TEACH presents the problem using the LED display and the voice synthesizer. The student's answer is entered via the keyboard. Two attempts to correctly answer the problem are permitted. If the answer is incorrect, two high tones are sounded and the display blinks. If the student's answer is correct, three low tones are produced by the synthesizer. Also, if the answer was correct or if two incorrect attempts have been made, TEACH will present both the problem and the correct answer to the student, in the form of a complete mathematical sentence.

TEACH keeps track of the number of questions that the student has been asked, the number of problems missed, and the total amount of time the student has used to solve the problems.

Depending on the option that is selected by the teacher, TEACH presents the problems in different ways (see Fig. 20). If the timed option is selected, the student has a fixed amount of time in which to work as many problems as possible. There is no fixed limit on the amount of time that can be spent on one given problem. If the speed option has been chosen, the student has a fixed amount of time for each attempt and a prescribed number of problems to do. The power option
FIGURE 19 - REPORT Subroutine
FIGURE 20 - TEACH Subroutine
FIGURE 20 - Continued
specifies that the student has a fixed number of problems to solve and an unlimited amount of time in which to do them.

2. Input/Output Subroutines

These routines control the information flow between the display, keyboard, and speech synthesizer via the microprocessor-controller.

a. BLANK Subroutine. BLANK (shown in Fig.21) stores a blank (FF₁₆) in all of the locations of the display buffer. The effect of BLANK is to turn off the display while SCAN is executed.

b. BLANKIT Subroutine. BLANKIT, which is shown in Fig.22, blanks the missing number in the display buffer. If a problem is to be displayed without the answer appearing on the LED's, the routine DISPLAY would be called to put the entire display into the buffer and BLANKIT would be called to blank the answer.

c. DISPLAY Subroutine. DISPLAY calls UNPACK to load the contents of NUM1, NUM2, and NUM3 into the corresponding display buffers (see Fig.23). The buffers for the equal sign and the operation sign are also loaded with their proper values. Leading zeros in the display are blanked by storing a blanking code (FF) in place of the zeros.

d. GETDIG Subroutine. GETDIG, shown in Fig.24, is used whenever a multidigit number is needed from the keyboard. GETDIG is entered with the time limit (in scans) stored in the X register (like SCAN). GETDIG returns when the time limit is exhausted, or if six digits have been entered. Pushing the enter key also causes a return from GETDIG. The entered digits are stored in a push down stack consisting of DIG1 through DIG6. The last digit entered is always stored at DIG1, and previous entries are pushed down toward DIG6. The total
FIGURE 21 - BLANK Subroutine

BLANK

A = $FF
X = starting
Addr. of Dis.
Buffer

((X)) = A
X = X + 1

N

X beyond
buffer?

RTS
FIGURE 22-BLANKIT Subroutine
UNPACK NUM1 into buffer for 1st display
UNPACK NUM2 into buffer for 2nd display
UNPACK NUM3 into buffer for 3rd display
Calculate value to turn on OPSIGN
Supress leading zeros in 1st display
Supress leading zeros in 2nd display
Supress leading zeros in 3rd display

SUPPRESS

(Display)

Turn on equals

Supress leading zeros in 1st display

Using SUPPRESS in subroutine in DISPLAY

FIGURE 23 - DISPLAY Subroutine
FIGURE 24 - GETDIG Subroutine
FIGURE 24 -Continued

GETDIG2

Inc. SGNDIG

Push down digits so last entry is at dig1, next to last is at dig2, etc.

X=TIMER

MORE

Y

Number of digits<6?

N

RTS

FIGURE 24 -Continued
number of digits entered is stored in SGNDIG. Pushing the minus key causes the sign bit (bit 7) of SGNDIG to be set. Pushing the repeat button causes a call to SAYDISP, and GETDIG is restarted (clearing any entered digits).

e. GETEM Subroutine. GETEM (see Fig.25) is a general purpose prompt and keyboard entry subroutine. The contents of the A-register are stored in the prompt buffer. GETEM then causes the words "GO PLEASE" to be loaded into the speech buffer. A call to GETDIG brings in up to 6 digits from the keyboard (see Fig. 24).

f. PACK Subroutine. PACK takes the information received from the keyboard by GETDIG and compacts it into a two-byte packed BCD number at the memory location addressed by X and X+1 (see Fig.25).

g. SAYDISP Subroutine. SAYDISP is the routine that loads the present problem into the speech buffer. Depending on the value of the variable SAY, one of three things will happen: 1) if SAY is equal to zero then SAYDISP will return with no action having been taken, 2) if SAY is equal to one, SAYDISP will load the problem into the buffer with a low tone in place of the missing number and 3) if SAY is equal to two, the entire problem including the missing number, is loaded into the speech buffer. The flowchart for SAYDISP is shown in Fig.26.

h. SCAN Subroutine. SCAN is the routine that controls all of the input and output functions for the device; it performs the keyboard scanning for input data, and multiplexes the data to the LED display. The desired number of SCANS (at 50 scans/sec) is stored in the X-register before calling SCAN. When SCAN is called, the
GETEM

Prompt=A reg.

Say "Go Please" (2BEEP)

Call GETDIG

PACK

A=DIG2
Shift A Right 4 times

OR A with DIG1

Store A at X + 1

Append Sign to DIG3 in A

Set sign bit if no digits were entered

do default values always wrong

RTS

FIGURE 25 - GETEM and PACK Subroutines
Load 2/5 sec. pause (LDPAUS)

SAY=2?

is number the missing number?

Call SAYNUM

RTS

Say "Lo-tone" (SPKWD)

RTS

SAYDISP

SAY=0 Y

RTS

SETUP for NUM1

Call CHKSPK

A=OPSIGN
Adjust A by adding $0E

Say OPSIGN by calling SPKWD

Call CHKSPK for NUM2

Say "EQUALS" (SPKWD)

Call CHKSPK for NUM3

Load ½ sec pause (LDPAUS)

RTS

FIGURE 26 - SAYDISP Subroutine
information stored in the display buffer (DISBUF) is routed to the display. SCAN also controls the speech synthesizer. Data stored in the speech buffer (SPKBUF) is sent to the synthesizer, one word at a time, until the speech buffer is empty. SCAN continues execution until one of two things happen: 1) if the desired number of scans has been completed, SCAN will return; and 2) depressing and releasing a key on the keyboard will cause a return from SCAN.

SCAN is divided into four major segments: SCAN DISPLAY, CHECK SPEECH, CHECK TIME, and KEY DEBOUNCE, that are executed sequentially.

SCAN DISPLAY controls the LED display and the keyboard. The display, shown in Fig. 10, plus the prompts are divided into 15 parts, each with its own four-bit device address. Corresponding to each device address is an output buffer location (in DISBUF). Also, each device address selects two keys as inputs. SCAN DISPLAY outputs the device addresses along with the data stored in DISBUF, and brings in the key information. A total is kept in SUM of the number of keys down simultaneously, if any. Also, the key number is stored in KEY (see Fig. 27). After SCAN DISPLAY has output to all 15 device addresses, control is passed on to CHECK SPEECH.

CHECK SPEECH, shown in Fig. 28, drives the TSI speech synthesizer. The synthesizer has 8 signal lines: six input data lines, one start line, and one busy status output line. When the start line is brought to the low state, the synthesizer begins saying the word selected by the six data lines. While the synthesizer is saying a word, its busy line is held in the low state. When the word is finished the busy
FIGURE 27 - SCAN DISPLAY Subroutine
NOTE: When busy goes high an interrupt brings START high.

FIGURE 28 - CHECK SPEECH Subroutine
line returns to the high state and causes an interrupt reentry to CHECK SPEECH.

CHECK SPEECH stores the data that is to be output to the synthesizer in a software FIFO (first-in, first-out buffer). The FIFO is organized as a circular buffer. Fig. 29 illustrates that buffer in two states, before and after data wrap around. Two pointers are used in this buffer; NEXT and LAST. NEXT always points to the next word in the buffer that is to be output. LAST Points to the location beyond the newest data word. When a new word is loaded into the buffer it is stored where LAST points and then LAST is incremented. In a similar way, when the word pointed to by NEXT has been spoken, NEXT is incremented by CHECK SPEECH to point to the next word in the buffer. If NEXT is incremented beyond the top of the buffer, it is set equal to the address at the bottom of the buffer. When NEXT equals LAST the buffer is empty.

The data in the buffer is of two types; words and pauses. A word is data that is destined to be routed to synthesizer. The six most significant bits of the word are sent to the synthesizer over the six data lines. The two least significant bits of a word must be zeros. If the two least significant bits of the buffer data are not zero, then the data is assumed to be a pause. A pause is an eight bit number representing the number of scans that should lapse before the next word is spoken. The maximum pause length is $2^8 - 1$ or 255 scans. At 50 scans per second the maximum PAUSE is 5.1 seconds.

When the device is initialized, the start line is set to its high state and NEXT and LAST are set equal. If SCAN is called at this point, the fact that NEXT equals LAST tells CHECK SPEECH that the
Circular FIFO Buffer before Data wrap around

Circular FIFO Buffer after Data wrap around

FIGURE 29 - Circular FIFO (Speech Buffer)
buffer is empty, and the rest of CHECK SPEECH is bypassed (see Fig. 28). When data has been stored in the buffer and SCAN is called, CHECK SPEECH is not bypassed. On the first pass through CHECK SPEECH, the data pointed to by NEXT is stored in the variable PAUSE if it is a pause and NEXT is INCREMENTED. If the data is a word to be spoken, it is put on the six data lines to the synthesizer and the PAUSE register is set to one. Further passes through CHECK SPEECH only decrements the PAUSE register. When the PAUSE register has been decremented to zero, and the data pointed to by NEXT is present on the data lines, CHECK SPEECH brings the start line from high to low. This triggers the synthesizer to begin saying the word. CHECK SPEECH also increments the value of NEXT, pointing it at the next data item. While the start line is low, CHECK SPEECH is bypassed. After the synthesizer has finished saying the word, it brings the busy line high, which causes an interrupt. The INTERRUPT routine brings the start line high, enabling CHECK SPEECH.

The buffer is 32 bytes long. It was felt that this length was large enough that the author would not have to be concerned about the program overflowing it. Also, having the audio output stored in a queue allows the device to simultaneously output speech, output to the LED display, and bring in key information.

The third major segment of SCAN is CHECK TIME. When SCAN is called, the number in the X-register is stored in SCNTIM. Normally, on each pass through CHECK TIME, SCNTIM is decremented. This occurs only if SCNTIM is not equal to its maximum value \(65,535_{10}\) (or \(FFFF\) in hexadecimal). When SCNTIM has been decremented to zero, a return is
performed and the computer exits SCAN. The maximum amount of time in SCAN is 21.84 minutes (65,535 scans). However, if SCAN is called with the X-register containing $FFFF_{16}$, SCAN will not automatically time out (see Figs. 30 and 31).

The KEY DEBOUNCE portion of SCAN takes the information provided by SCAN DISPLAY and uses it to determine when a key has been pushed and released. For a key to be valid, it must be held down for two scans of the keyboard and then released. If more than one key is depressed, then the keys are ignored. This technique of debouncing is often used in calculators. KEY DEBOUNCE is a software emulation of a finite state machine (see Figs. 30 and 31). When a key is released KEY DEBOUNCE causes a RTS to be executed.

i. **SPKEY Subroutine.** The SPKEY subroutine interprets the data in the A-register as a key number. If the key has a corresponding word in the synthesizer's vocabulary, that word is loaded into the speech buffer; otherwise, a high tone is loaded into the speech buffer (see Fig. 32). The keys that have corresponding words are zero through nine and the minus key.

j. **SPKWD and LDPAUS Subroutines.** SPKWD and LDPAUS are the subroutines that load data into the speech buffer. SPKWD is the routine that is used to load a word, stored in the A-register, into the buffer. LDPAUS takes the number in the A-register and stores it in the buffer. The routine calling LDPAUS must insure that one of the two least significant bits is a one, if SCAN is to interpret the number as a pause. After the data is inserted into the buffer, at the location pointed to by LAST, LAST is incremented so that it points to the
FIGURE 30 - CHECK TIME & KEY DEBOUNCE
Subroutine
FIGURE 31 - Finite State and Key Debounce
FIGURE 32 - SPKEY Subroutine
next empty buffer location (see Fig. 33).

k. UNPACK Subroutine. Decimal numbers, internal to the device, are stored in packed BCD format of the form:

\[
[SXXX, \text{hunds}]:[\text{tens, ones}]
\]

high byte low byte

where [ ] is a byte,

- \(S\) is a sign bit (1 implies minus),
- \(X\) is a don't care bit,
- \(\text{hunds}\) is the hundreds digit,
- \(\text{tens}\) is the tens digit,
- \(\text{ones}\) is the units digit.

UNPACK takes the number stored in A:B and puts this data, one digit per byte, into four bytes starting at the memory location addressed by the X-register. Because the routine is used to load the display buffer, the digits are stored in the high nibbles of the unpacked data bytes (see Fig. 34).

1. WAIT Subroutine. WAIT performs a function very similar to its name. WAIT continues to SCAN the display and output words to synthesizer until the speech buffer is empty. WAIT holds up program execution while the speech buffer is being emptied; it prevents the speech from getting too far behind the programs that generate the speech (see Fig. 35).

3. Arithmetic Subroutines

These routines perform the necessary background computations to generate problems for the student.
FIGURE 33 - SPKWD and LDPAUS Subroutines
UNPACK

Push A&B

Mask off Low Nibble of A

Store A at X+0

Pull A

Move Lo Nibble of A to High Nibble

Store A at X+1

Mask off Lo Nibble of B

Store B at X+2

Pull A

Move Lo Nibble of A to High Nibble

Store A at X+3

RTS

FIGURE 34 - UNPACK Subroutine
FIGURE 35 - WAIT Subroutine
a. **ADDEM Subroutine.** ADDEM is a signed BCD addition routine. It adds the signed BCD numbers in UNE and DEUX and places the result in TROIX.

ADDEM uses 10's complement arithmetic to add negative numbers (see Fig.36). If one of the numbers is negative, the 10's complement of that number is formed. The numbers are then added. If the result is negative, the ten's complement of the answer is taken and the sign bit is set.

b. **BCDADD Subroutine.** The BCDADD subroutine takes the two-byte BCD data word stored in A:B and decimal adds it to the contents of the location addressed by the X-register (see Fig.37).

c. **BCDINC Subroutine.** The BCDINC subroutine loads A:B with one and jumps to BCDADD. The net effect is to increment (BCD) the contents of the locations addressed by the X-register (see Fig.38).

d. **BCDMULT Subroutine.** BCDMULT is a signed BCD multiplication routine. The two numbers to be multiplied are in stored in packed BCD notation. The numbers are located in the double byte variables NUM1 and NUM2 and the result is stored in NUM3. NUM1 and NUM2 are not destroyed in the multiplication process. BCDMULT requires that NUM1 and NUM2 be between -99 and +99, and the resulting product must lie in the range of -999 to +999. A flow chart of BCDMULT appears in Fig.39.

e. **BISUB Subroutine.** The BISUB subroutine is a general purpose double byte binary subtraction routine. When called, it subtracts the number in A:B from the number addressed by X and X+1 (see Fig.40).
Subtract no. at x from 9999 to get g's comp.

Call BCDINC to form 10's complement

RTS

Negate

Take Absolute Value of UNE and DEUX

A:B=UNE TROIX=DEUX X=ADDR of TROIX

Call BCDADD

Recover Original Sign

Transfer Sign to Answer

RTS

ChkNeg

No. at x Negative?

RTS

Mask off Sign bit

Jump to Negate

Addem

Correct UNE and DEUX if they contain minus form

Are Signs Same?

Y

X=ADDR of UNE Call CHKNEG

X=ADDR of DEUX Call CHKNEG

TROIX=DEUX A:B=UNE X=ADDR of TROIX

Call BCDADD

No

Answer is Positive?

Y

Call Negate

Set Sign bit of TROIX

RTS

N

RTS

Figure 36 - Addem Subroutine
FIGURE 37 - BCDADD Subroutine

FIGURE 38 - BCDINC Subroutine
BCDMULT

UNE=NUM1
DEUX=NUM2

Clear sign of UNE
Clear TROIX

A=DEUX(LO)
Mask off hi nibble of A

ADD UNE to TROIX A times (BCDADD)

A=DEUX(LO)

Shift UNE:UNE+1
Left 4 times
Shift tens to ones in A

ADD UNE to TROIX A times (BCDADD)

Sign of TROIX=
Sign of NUM1 @
Sign of NUM2

NUM3=TROIX

FIGURE 39 - BCMULT Subroutine
f. **COMPARE Subroutine.** The need to be able to compare signed BCD numbers prompted the writing of COMPARE. COMPARE subtracts the number in the X-register from the variable VALUE using signed BCD arithmetic. The result is returned in the X-register (see Fig.41). Branching can be reliably done using the Z and N bits in the condition code register.

g. **DADD Subroutine.** DADD is a double byte binary addition routine. DADD adds the contents of A:B to the contents of the memory locations addressed by X and X+1. (See Fig.42).

h. **GETANS Subroutine.** GETANS is the routine that finds the answer to the problem defined by NUM1, NUM2, and OPSIGN. If, for example, OPSIGN was multiplication, then GETANS would call BCDMULT to find the product of NUM1 and NUM2 (see Fig.43). The result of the multiplication would be stored in NUM3.

GETANS doesn't always perform the operation suggested by OPSIGN. If OPSIGN corresponds to division, GETANS will multiply the contents of NUM1 and NUM2. The resulting product is moved to NUM1 and the original NUM1 and NUM2 are moved to NUM2 and NUM3 respectively. These moves correspond to forming the complementary division problem of the multiplication of NUM1 times NUM2.

Normally, subtraction is accomplished in GETANS by reversing the sign of the second number, and adding the two operands. However, if MODE corresponds to Level of Difficulty and bit 14 of PROBTYPE (called the T-bit) is set, then subtraction is done by adding NUM1 and NUM2. The sum is stored at NUM1 and the original NUM1 and NUM2 are moved to NUM2 and NUM3, respectively. This
COMPARE

DEUX=X

UNE=Value

Complement Sign of DEUX

Call ADDEM

X=TROIX

RTS

FIGURE 40 - COMPARE Subroutine

BISUB

Take 2's complement of A:B

Jump to DADD

FIGURE 41 - BISUB Subroutine
FIGURE 42 - DADD Subroutine

- DADD
- \( B = B + ((X + 1)) \)
- \( ((X + 1)) = B \)
- \( A = A + ((X)) + C \), where \( C = \text{carry from previous addition} \)
- \( ((X)) = A \)
- RTS
Figure 43 - GETANS Subroutine
produces the complementary subtraction problem to the problem, NUM1 plus NUM2.

i. HI-LO Subroutine. HI-LO compares the contents of VALUE and the upper and lower bounds imbedded in the code of the calling routine (see Figs. 44 and 45); if VALUE is between the limits then the carry bit is set, otherwise it is cleared.

HI-LO calls LOADX to retrieve the inline data and to increase the return address stored on the stack so that when the RTS instruction is executed, the program counter points to the next instruction in the calling routine, and not to the data. HI-LO calls COMPARE to determine if VALUE is between the limits.

j. RANDOM Subroutine. The RANDOM subroutine generates a random number using the two byte seed addressed by the X-register. The new seed is stored at the two memory locations addressed by X. The resulting random number is stored in the two byte variable VALUE.

The seed is treated as a linear feedback shift register. The seed is shifted left and the new bit zero is formed by Exclusive OR'ing the previous bit zero and bit ten together (see Fig. 46). This feedback register has a period of $2^{11}-1 (=2045)$. The seed is Exclusive OR'ed with a mask to further randomize the number. The random number is then converted from binary to decimal and stored in VALUE.

4. Special Function Subroutines

This section contains the descriptions of subroutines that perform special functions for the control subroutines.
Calling routine

DATA

- JSR HI-LO

\{ 
  - hilim
  - lolim
  - NEXT inst.
\}

HI-LO

Set C if Lo\text{lim}<\text{VALUE} \leq hi\text{lim}

RTS

SPECIAL RETURN PATH

NORMAL RETURN

FIGURE 44 - How HI-LO Works
FIGURE 45 - HI-LO Subroutine
Let \text{VALUE}=-421

Check for zero in bits 0 to 10 of Mem@X

Hi Byte @5A5, Push onto stack
Lo Byte @5A5, A=Lo Byte

Clear Running total (\text{VALUE})
X=Starting ADDR or random

BSR CONVERT

Pull A from STACK

BSR CONVERT

Let \text{VALUE}=-421

\text{VALUE}=\text{min}us zero?

RTS

\text{FIGURE 46 - RANDOM Subroutine}
a. **ERROR Subroutine.** The ERROR subroutine causes the display to blink several times and the speech synthesizer to output two high tones (see Fig.47). ERROR is used as negative feedback by TEACH when the student enters an incorrect answer.

b. **GENERATE Subroutine.** The GENERATE subroutine, using the information provided by the teacher, randomly chooses the values of the two operands, NUM1 and NUM2. In the level of difficulty mode, GENERATE calls OPGEN to obtain the values for NUM1 and NUM2. Also, if regrouping is not allowed, GENERATE insures that is is not present in the problem (see Fig.48). If the Range mode is selected, GENERATE insures that NUM1 is between HILIM1 and LOLIM1, and that NUM2 is between HILIM2 and LOLIM2.

c. **INTERRUPT Program.** When the speech synthesizer is through talking, its Busy lines goes low, and imitates an interrupt. When an interrupt is received, the interrupt program sets the synthesizer's start line to its high state (see Fig.49) and thereby informs SCAN that the synthesizer has completed talking. After the interrupt has been serviced, the computer resumes what it was doing previously.

d. **LINK.** LINK is neither a program nor a subroutine. LINK provides a way of tying together all of the subroutines used in the device. All subroutine calls are passed by jump statements in LINK to the called subroutines. This allows a subroutine to be moved in memory with very little trouble. Normally, when a subroutine is moved, all of the calls to that subroutine must be changed. With LINK all of the calls to a particular subroutine actually call a jump statement in LINK; the jump statement then transfers control to the called
ERROR

Call WAIT

DIG5=1
DIG6=$0A

Say "Hi Tone" twice (using SPKWD)

Dec. DIG5

DIG5=∅?

Call BLANK

Call BLANKIT

SCAN for 1/10 sec

DEC DIG6

SCAN for 1/10 sec

Call DISPLAY

DIG6=∅?

RTS

FIGURE 47 - ERROR Subroutine
FIGURE 49 - INTERRUPT Subroutine

INTERRUPT

Set Start to 1

Reset Interrupt bit in PIA

RTI
subroutine. Thus, when the called routine is moved, only the jump address in LINK must be changed.

e. **LOADX Subroutine.** LOADX passes parameters from a primary routine to a secondary routine. The two byte parameter is embedded in the primary routine immediately after the call to the secondary routine (Figs. 50 and 51). The secondary routine calls LOADX, which loads the parameter into the X-register. LOADX also adjust the primary routine's return address located on the stack, so that a RTS in the secondary routine causes a return to the primary routine beyond the parameter. LOADX preserves the values of registers A and B by storing them on the stack.

f. **MINTOBI Subroutine.** The MINTOBI subroutine derives its name from MINutes TO BIinary. This subroutine assumes that GETDIG has been called previously and that a number of the form, [tens of minutes, unit minutes, tens of seconds, unit seconds], has been entered. MINTOBI converts this data to binary seconds and stores it at the two memory locations addressed by the X-register (see Fig.52). MINTOBI does this by multiplying the digits 1 through 4 by weighting factors.

g. **OPGEN Subroutine.** The OGEN subroutine generates a random number within a certain range. This range varies with the contents of the A-register when OGEN is called (see TABLE VII). The Seed used for the random number is stored at the memory locations addressed by the X-register when OGEN is called.

OPGEN calls RANDOM to generate the random number (see Fig.53). The number is then adjusted to fit the limits defined by A (i.e. if the
FIGURE 50 - How LOADX Works
FIGURE 51 - LOADX Subroutine
MINTOBI

Clear running total at X

Store DIG1 in total (Lo)

Add $10^{10} \times DIG2$ to total (DADD)

Add $60 \times DIG3$ to total (DADD)

Add $600 \times DIG4$ to total (DADD)

RTS

FIGURE 52 - MINTOBI Subroutine
### TABLE VIII

**OPERAND RANGE AS DEFINED BY A-REGISTER**

<table>
<thead>
<tr>
<th>Contents of A(hex)</th>
<th>Range of Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 to +4</td>
</tr>
<tr>
<td>1</td>
<td>-4 to +4</td>
</tr>
<tr>
<td>2</td>
<td>+5 to +9</td>
</tr>
<tr>
<td>3</td>
<td>-9 to -5 and +5 to +9</td>
</tr>
<tr>
<td>4</td>
<td>0 to +9</td>
</tr>
<tr>
<td>5</td>
<td>-9 to +9</td>
</tr>
<tr>
<td>6</td>
<td>0, 10, 20,..., 90</td>
</tr>
<tr>
<td>7</td>
<td>-90, -80, -70,..., 70, 80, 90</td>
</tr>
<tr>
<td>8</td>
<td>+10 to +99</td>
</tr>
<tr>
<td>9</td>
<td>-99 to -10 and +10 to +99</td>
</tr>
<tr>
<td>A</td>
<td>-99 to +99</td>
</tr>
<tr>
<td>B</td>
<td>+100 to +499</td>
</tr>
<tr>
<td>C</td>
<td>-499 to -100 and +100 to +499</td>
</tr>
<tr>
<td>D</td>
<td>-499 to +499</td>
</tr>
<tr>
<td>E</td>
<td>Same as D</td>
</tr>
<tr>
<td>F</td>
<td>Same as D</td>
</tr>
</tbody>
</table>
FIGURE 53 - OPGEN Subroutine
number is defined as two digits, then the third digit is blanked). If the number is still outside the defined limits, another random number is generated and the limits are checked again.

h. RIGHT Subroutine. The RIGHT subroutine is used by TEACH as positive feedback for the student. When called, RIGHT turns off the prompt and sounds a low tone three times. A flowchart of RIGHT appears in Fig. 54.

i. RESTART Program. The RESTART program is called on device power-up or when the reset button is pushed. RESTART initializes the PIA's and it initializes the variable that controls scan speed (RATE), the display buffer (DISBUF), the pointers for the speech buffer (NEXT and LAST) (see Fig. 55). The processor's stack pointer is also initialized to its starting position at $00FF$. After RESTART is executed, control is passed to MAIN.

j. SAYNUM Subroutine. The SAYNUM subroutine takes the number addressed by the X-register and says it. The number is assumed to have the following two byte format:

[SXXX, hundreds]:[tens, ones]

at (X)+ at (X+1)+

where;  S is the sign bit (set implies minus)
Z is a "don't care" bit
hundreds, tens, and ones are stored in BCD notation
RIGHT

Turn Off Prompt

Say "LO TONE" Three times (using SPKWD)

Call WAIT

RTS

FIGURE 54 - RIGHT Subroutine
RESTART

Load Delay Constant

Initialize Stack Pointer

Specify Data dir Reg. in PIA

Load Dir Reg's

Initialize Control Reg A
Initialize Control Reg B (start=1)

Initialize NEXT & LAST

Blank DISBUF

To MAIN

FIGURE 55 - RESTART Subroutine
(See Figure 56 for the exact algorithm used to say the number).

SAYNUM says the numbers as the students are taught to say them. For example, -327 is pronounced as "minus three hundred twenty-seven," not as "minus three two seven."

k. TBLOOK Subroutine. TBLOOK is the subroutine that looks up the problem set information when the device is in the Level of Difficulty mode. The problem set information is contained in three different tables. If the operation is addition, then the problem set information is in ADDTBL. If the operation is subtraction then SUBTBL contains the information. Multiplication and division information is stored in M-DTBL. All of the tables end with the flag $FFFF. If the level of difficulty exceeds the number of problem sets (and the flag is found), TBLOOK will cause the device to restart (see Fig. 57).

l. 2BEEP Subroutine. The 2BEEP subroutine is an audio prompting subroutine. Originally call 2BEEP would cause two "hi tones" to be sounded, hence its name. 2BEEP was changed so that calling it causes the words "Go Please" to be loaded into the speech buffer (see Fig. 58).
FIGURE 56 - SAYNUM Subroutine
Using indexed addressing load XSAV with starting Address of:
ADDTBL if OPSIGN=Add
SUBTBL if OPSIGN=Sub.
M-DTBL if OPSIGN=Mul or Div

PROBTYPE=$0001
(this is BCD Counter)

INC(BCD) PROBTYPE (BCDINC)

X=((XSAV))

X=$FFFF?
Y
Jump to RE-START

XSAV=XSAV+2

PROBTYPE = LEVEL?
Y

PROBTYPE=((XSAV))

FIGURE 57 - TBLOOK Subroutine
FIGURE 58 - 2BEEP Subroutine
VI. CONCLUSIONS AND RECOMMENDATIONS

A. Cost Analysis

The Math Trainer was designed to be a proof-of-concept prototype. As such, it was not designed to be mass produced in its present form. This version is too expensive to be marketable in reasonable quantity. After evaluating the device's usefulness and implementing any necessary changes, use of production oriented technology could render it more marketable. A breakdown of costs for the components of the device is shown in TABLE IX. Not shown in TABLE XI are the development costs and the cost of the labor required to assemble the device.

B. Device Improvement Recommendations

The Math Trainer works as it was designed to. The lack of funds to install solenoids as prompts has left the device somewhat difficult, but not impossible, for a blind teacher to program. If such funds become available in the future, the installation of solenoids is highly recommended.

It was noted that some users find the order of programming questions unnatural. Rearranging the question order would be a simple matter and, as such, may be desirable. If PROGRAM were to be rewritten, a useful addition would be to implement error checking of the numeric inputs (preventing negative numbers entered as a position, for example). The trainer could be made more compact by using a CMOS microprocessor and a single voltage 4K EPROM. Doing that
<table>
<thead>
<tr>
<th>Description</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech Synthesizer</td>
<td>$189.00</td>
</tr>
<tr>
<td>Support Circuits for Synthesizer</td>
<td>70.00</td>
</tr>
<tr>
<td>D1 Microcomputer</td>
<td>75.00</td>
</tr>
<tr>
<td>(4) 2708 EPROMS</td>
<td>48.00</td>
</tr>
<tr>
<td>Cabinet for Device</td>
<td>100.00</td>
</tr>
<tr>
<td>Power Supply</td>
<td>50.00</td>
</tr>
<tr>
<td>(13) DL747 LED Displays</td>
<td>13.00</td>
</tr>
<tr>
<td>Keyboard Components</td>
<td>15.00</td>
</tr>
<tr>
<td>Miscellaneous Electronic Components</td>
<td>75.00</td>
</tr>
<tr>
<td>Miscellaneous Hardware</td>
<td>40.00</td>
</tr>
<tr>
<td><strong>TOTAL COST</strong></td>
<td><strong>$675.00</strong></td>
</tr>
</tbody>
</table>
would allow the bulky multi-voltage supply to be replaced by a much smaller power supply.

Currently, the device is slated to be evaluated by using it at the Kentucky School for the Blind in the Fall of 1979. Some changes may be suggested by that evaluation.

C. Notes on Doing Similar Projects

Several things became apparent while working on the Math Trainer. The creation of the device at the assembly language level made the device difficult to change. Any feedback from users was difficult to take advantage of due to the difficulty of changing existing programs. Anyone working on a similar project would be prudent to insure that the man-machine interface design was finalized before writing the control software. For some projects, a high level language program (e.g. in BASIC) could be written to simulate the device. Changes could be easily made to the simulation program in response to user feedback. These changes could then be incorporated into the machine language control program from the beginning. The use of a "Tiny Basic" may even allow the final control program to be written in a high level language.

An assembler would also be a great aid in producing control programs. Although an assembler was made available to the author it was not used because most of the software was written by the time the assembler was obtainer. A great deal of time was spent by the author debugging incorrectly hand-assembled programs.
One resource that was not taken full advantage of was the extensive amount of arithmetic software already written for the 6800. The author's experience indicates that it would be worthwhile to use some of this available software even if some data conversion were required.
REFERENCES CITED


2. Brothers, Roy J., Arithmetic Computation: Achievement of Visually Handicapped Students in Public Schools, American Printing House for the Blind (September, 1972), Louisville, KY.


4. Brothers.


8. Obtained from Quota Registrations (as of Jan. 5, 1976), provided by J. E. Morris of American Printing House for the Blind, Louisville, KY.

9. "Speak and Spell" pamphlet, Texas Instruments, PO 225012, Dallas, TX 75295.

10. Telesensory Systems, Inc.


12. Personal communication with Dr. Charles S. Thompson, Elementary Education, University of Louisville.

13. Ibid.

14. Ibid.

15. Texas Instruments, PO 5474, Dallas, TX 75222.

17. Ibid.

18. Ibid.


20. Litronix, Inc., 19000 Homestead Road, Vallco Park, Cupertino, California 95014.

21. Telesensory Systems, Inc.

22. Ibid.

23. Motorola Semiconductor Group, 5005 E. McDowell Road, Phoenix, Arizona 85008.

24. Ibid.


26. Pro-Log Corporation, 2411 Garden Road, Monterey, California 93940.

27. Litronix, Inc.


29. Haviland, R. P.

BIBLIOGRAPHY


11. Texas Instruments, "Speak and Spell" pamphlet, Texas Instruments, PO 225012, Dallas, TX 75295.


14. Personal Communication with Dr. Charles S. Thompson, Elementary Education, University of Louisville.
APPENDIX I

DEVELOPMENT CIRCUITRY
FIGURE 1 - D2 Board Microcomputer Module
FIGURE 1 - D2 Board Microcomputer Module (continued)
FIGURE 1 - D2 Display Module
FIGURE 1 - D2 Display Module (continued)
FIGURE 2 - 4K RAM Board
FIGURE 3 - Memory Protect Circuit
APPENDIX II

DEVICE HARDWARE DOCUMENTATION
FIGURE 1 - Continued
FIGURE 2 - Display Board (Section 1)
FIGURE 2 - Display Board Section 2 (LED Display)
FIGURE 2 - Display Board Section 3 (Prompts)
FIGURE 2 - Display Board Section 4 (Operation Sign)

NOTE: Q14 - Q17
2N3906
FIGURE 2 - Display Board  Section 5 (Front View)
FIGURE 3 - EPROM Board (Back View)
FIGURE 4 - Keyboard (Back View)
FIGURE 5 - Jumper Board (Back View)
FIGURE 6 - Power Supply
<table>
<thead>
<tr>
<th>Connector</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connector C1</td>
<td>maps directly to pins on C7.</td>
</tr>
<tr>
<td>Connector C2</td>
<td>maps directly to pins on C8.</td>
</tr>
<tr>
<td>Connector C3</td>
<td>maps directly to pins on C9.</td>
</tr>
<tr>
<td>Connector C5</td>
<td>maps directly to pins on C6.</td>
</tr>
</tbody>
</table>

**CONNECTOR C7**

Pins 4 through 11 are GND.  
Other pins not connected.

**CONNECTOR C8**

Pin 1 = C11,1  
2 = C11,13  
3 = C11,2  
4 = C11,12  
5 = C11,3  
6 = C11,11  
7 = C11,4  

Pin 8 = C10,2  
9 = C10,13  
10 = C10,6  
11 = C10,9  
12 = C10,5  
13 = C10,10  
14 = C10,4

**CONNECTOR C9**

Pin 1 = +5v  
2 = +5v  
3 = +5v  
4 = +5v  
5 = C10,8  
6 = C10,7  
7 = C11,14  

Pin 8 = C10,11  
9 = C10,13  
10 = C10,12  
11 = +5v  
12 = +5v  
13 = +5v  
14 = Not Connected

**CONNECTOR C4**

Pin 1 = Pin 9 on SPEECH BOX  
2 = Pin 10 on SPEECH BOX  
3 = Pin 11 on SPEECH BOX  
4 = Pin 12 on SPEECH BOX  
5 = Pin 13 on SPEECH BOX  
6 = Pin 14 on SPEECH BOX  
7 = Pin 15 on SPEECH BOX  

Pin 8 = Pin 1 on SPEECH BOX  
9 = Pin 2 on SPEECH BOX  
10 = Pin 3 on SPEECH BOX  
11 = Not Connected  
12 = Not Connected  
13 = Not Connected  
14 = Not Connected
<table>
<thead>
<tr>
<th>EPROM CONNECTOR</th>
<th>D1 CONNECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>10</td>
</tr>
<tr>
<td>T</td>
<td>40</td>
</tr>
<tr>
<td>S</td>
<td>V</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>P</td>
<td>39</td>
</tr>
<tr>
<td>N</td>
<td>38</td>
</tr>
<tr>
<td>M</td>
<td>T</td>
</tr>
<tr>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>K</td>
<td>37</td>
</tr>
<tr>
<td>J</td>
<td>36</td>
</tr>
<tr>
<td>H</td>
<td>R</td>
</tr>
<tr>
<td>F</td>
<td>P</td>
</tr>
<tr>
<td>E</td>
<td>35</td>
</tr>
<tr>
<td>D</td>
<td>34</td>
</tr>
<tr>
<td>C</td>
<td>N</td>
</tr>
<tr>
<td>B</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>N/C</td>
</tr>
<tr>
<td>9</td>
<td>N/C</td>
</tr>
<tr>
<td>10</td>
<td>N/C</td>
</tr>
<tr>
<td>11</td>
<td>RESTART SWITCH</td>
</tr>
<tr>
<td>12</td>
<td>1,2,3,A,B,C</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>W,X,Y,41,42,43</td>
</tr>
<tr>
<td>15</td>
<td>N/C</td>
</tr>
<tr>
<td>16</td>
<td>J</td>
</tr>
<tr>
<td>17</td>
<td>L</td>
</tr>
<tr>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>32</td>
</tr>
<tr>
<td>20</td>
<td>H</td>
</tr>
<tr>
<td>21</td>
<td>K</td>
</tr>
<tr>
<td>22</td>
<td>29</td>
</tr>
<tr>
<td>23</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PIA CONNECTOR</th>
<th>CONNECTED TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>C10,13</td>
</tr>
<tr>
<td>37</td>
<td>C10,2</td>
</tr>
<tr>
<td>36</td>
<td>C10,12</td>
</tr>
<tr>
<td>35</td>
<td>C10,3</td>
</tr>
<tr>
<td>34</td>
<td>C10,11</td>
</tr>
<tr>
<td>33</td>
<td>C10,4</td>
</tr>
<tr>
<td>32</td>
<td>C10,10</td>
</tr>
<tr>
<td>31</td>
<td>C10,5</td>
</tr>
<tr>
<td>30</td>
<td>C10,9</td>
</tr>
<tr>
<td>29</td>
<td>C10,6</td>
</tr>
<tr>
<td>28</td>
<td>C10,8</td>
</tr>
<tr>
<td>27</td>
<td>C10,7</td>
</tr>
<tr>
<td>26</td>
<td>C11,14</td>
</tr>
<tr>
<td>25</td>
<td>C11,1</td>
</tr>
<tr>
<td>PIA CONNECTOR</td>
<td>CONNECTED TO</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Pin 24</td>
<td>C11,13</td>
</tr>
<tr>
<td>23</td>
<td>C11, 2</td>
</tr>
<tr>
<td>22</td>
<td>C11,12</td>
</tr>
<tr>
<td>21</td>
<td>C11,3</td>
</tr>
<tr>
<td>20</td>
<td>C11,11</td>
</tr>
<tr>
<td>19</td>
<td>C11,4</td>
</tr>
</tbody>
</table>
APPENDIX III
SOFTWARE DOCUMENTATION

These listings were produced by using a disassembler program on a D2 microcomputer. The disassembled listings, output by the D2, were transferred to the University of Louisville's DEC-10 computer via a digital cassette recorder. Using the DEC-10's text editors (SOS and TECO), comments and symbolic addresses were added. The listings were printed on a DEC-WRITER. The SOS editor was also used to generate the subroutine hierarchy listings.
TABLE I

SYMBOL TABLE

Stack starts at $00FF

PIA Addresses

$8008 = PORT-A
$8009 = CONT-A
$800A = PORT-B
$880B = CONT-B

RAM Addresses

$0000 = KEY
$0001 = BF
$0002 = DADDR
$0003 = SUM
$0004-04 = XSAV
$0006-15 = DISBUF
$0016-17 = NEXT
$0018-19 = LAST
$001A-1B = SCNTIM
$001C = PAUSE
$001D-3C(32 bytes)=SPKBUF
$003D = SGNDIG
$003E-43 = DIG1-DIG6
$0044-45 = TIMER
$0047-47 = RATE
$0048 = MODE
$0049 = OPTION
$004A-4B = SEED 1
$004C-4D = TIMLIM
$004E-4F = MAX PROBS
$0050 = OPSIGN
$0051-52 = MISSING #
$0053-54 = HILIM1/LEVEL
$0055-56 = PROTYPE/LOLIM1
$0057-58 = HILIM2
$0059-5A = LOLIM1
$005B-5C = NUM1
$005D-5E = NUM2
$005F-60 = NUM3
$0061 = SAY
$0062-63 = SEED 2
$0064-65 = ATTPROBS
$0066-67 = WRONG
$0068-69 = SECONDS
$006A-6B = MINUTES
$006C = ERROR
$006D-6E = DISPTIME
$006F-70 = ANSWER
$0071-72 = BINSECS
$0075-76 = VALUE
$0077-78 = UNE
$0079-7A = DEUX
$007B-7C = TROIX
$007D-7E = SEED A
$007F-80 = SEED B
$0081-82 = ADDRESS

ROM MEMORY ADDRESS

$2000-2026 = RESTART(1)
$2027-202F = INTERRUPT
$2030-2045 = RESTART(2)
$2046-2081 = MAIN
$209F-20FC = RANDOM
$211D-21C9 = OPGEN
$21CA-21E4 = HI-LO
$2200-222F = DOWN(utility)
$2240-228F = DISPLAY
$22C7-22D5 = BCDADD
$22D6-22DA = BCDINC
$22D6-22DA = LOADX
$22F4-241D = PROGRAM
$2455-2467 = 2BEEP
$2469-24A6 = MINTOBI
$24A7-24FF = SAYNUM
$2500-25FD = SCAN
$2600-2611 = SPKWD and LDPAUS
$2612-261D = GETEM
$2620-262F = SPKEY
$2634-2670 = DRILL
$2680-268C = WAIT
$268D-2702 = GETDIG
$2710-272F = BLANKIT
$2730-2748 = UNPACK
$2744-2798 = SAYDISP
$279A-2810 = LINK
$281A-2868 = BCDMULT
$2869-28FF = GETANS
$2910-2922 = BISUB
$2924-2935 = COMPARE
$2936-2A3A = GENERATE
$2A43-2ABD = REPORT
$2ABE-2BE3 = TEACH
$2C00-2C3A = ERROR
$2C40-2CB9 = ADDEM
$2CBF-2CC7 = DADD
$2CCF-2D16 = TBLOOK
$2E00-2E3A = ADDTBL
$2E80-2EA8 = SUBTBL
$2F10-2F19 = M-DTBL
$2F70-2F7F = BLANK
$2FAB-2FC8 = PACK
$2F80-2F96 = RIGHT
$2FF8-2FFF = RESTART AND INTERRUPT VECTORS
## TABLE II

**SUBROUTINE HIERARCHY**

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2024</td>
<td>7E 2030 JMP $2030</td>
<td><strong>START OF RESTART PROGRAM</strong></td>
</tr>
<tr>
<td>2026</td>
<td>BD 27D2 JSR PROGRAM</td>
<td>* TEACHER DEFINES PROB. SETS</td>
</tr>
<tr>
<td>2029</td>
<td>BD 27ED JSR DRILL</td>
<td>* KEYBOARD TRAINING DRILL</td>
</tr>
<tr>
<td>2032</td>
<td>BD 27A0 JSR TBLOOK</td>
<td>* LOOK UP PROBLEM SET INFO</td>
</tr>
<tr>
<td>2036</td>
<td>BD 279A JSR TEACH</td>
<td>* TEACH MATH PROBLEMS</td>
</tr>
<tr>
<td>2039</td>
<td>BD 27FC JSR REPORT</td>
<td>* GIVE TOTAL RIGHT &amp; TIME</td>
</tr>
<tr>
<td>2042</td>
<td>BD 27C6 JSR BLANK</td>
<td>* BLANK ENTIRE DISPLAY</td>
</tr>
<tr>
<td>2045</td>
<td>BD 27F6 JSR SPKWD</td>
<td>* LOAD WORD INTO SPEECH BUF.</td>
</tr>
<tr>
<td>2048</td>
<td>BD 27F9 JSR SCAN</td>
<td>* INPUT:KEYS OUTPUT:DISP. &amp; TALK</td>
</tr>
<tr>
<td>2051</td>
<td>7E 204C JMP $204C</td>
<td><strong>END OF MAIN</strong>**</td>
</tr>
<tr>
<td>2054</td>
<td>BD 27B4 JSR BCDADD</td>
<td>* ADD A:B TO LOC @X</td>
</tr>
<tr>
<td>2057</td>
<td>BD 2784 JSR BCDTBL</td>
<td><strong>START OF BCDTBL LOOK UP TABLE</strong>**</td>
</tr>
<tr>
<td>2060</td>
<td>BD 2785 JSR BCDTBL</td>
<td><strong>END OF BCDTBL</strong>**</td>
</tr>
</tbody>
</table>
**** START OF UPGEN SUBROUTINE ****
2131 BD 27AC JSR RANDOM
2134 BD 27FF JSR HI-LO
2162 BD 27FF JSR HI-LO
2191 BD 27FF JSR HI-LO
21A6 BD 27FF JSR HI-LO
**** START OF HI-LO SUBROUTINE ****
21CA BD 2802 JSR LOADX
21CD BD 279D JSR COMPARE
21D4 BD 2802 JSR LOADX
21D9 BD 2802 JSR LOADX
21DC BD 279D JSR COMPARE
**** START OF DOWN UTILITY PROGRAM ****
**** START OF DISPLAY SUBROUTINE ****
224B BD 27CC JSR UNPACK
2245 BD 27CC JSR UNPACK
225F BD 27CC JSR UNPACK
**** START OF UP UTILITY PROGRAM ****
**** START OF BCDADD SUBROUTINE ****
**** START OF BCDINC SUBROUTINE ****
22D9 7E 27B4 JMP BCDADD
**** START OF LOADX SUBROUTINE ****
**** START OF PROGRAM SUBROUTINE ****
230B BD 27E1 JSR 2BEEP
2311 BD 27F9 JSR SCAN
2316 BD 27F0 JSR SPKEY
232D BD 27E1 JSR 2BEEP
2333 BD 27F9 JSR SCAN
2338 BD 27F0 JSR SPKEY
2356 BD 27E1 JSR 2BEEP
235C BD 27E7 JSR GETDIG
* RANDOM NUMBER GENERATOR
* SET C IF VALUE BETWEEN LIMITS
* SET C IF VALUE BETWEEN LIMITS
* SET C IF VALUE BETWEEN LIMITS
* SET C IF VALUE BETWEEN LIMITS
* LOAD X FROM INLINE DATA
* COMPARE (BCD) VALUE & X
* LOAD X FROM INLINE DATA
* LOAD X FROM INLINE DATA
* COMPARE (BCD) VALUE & X
* SEPERATE NIBBLES INTO BYTES
* SEPERATE NIBBLES INTO BYTES
* SEPERATE NIBBLES INTO BYTES
* ADD A:B TO LOC @ X
* SAY "GO PLEASE"
* INPUT KEYS OUTPUT DISP. & TALK
* ECHO KEY NAME
* SAY "GO PLEASE"
* INPUT KEYS OUTPUT DISP. & TALK
* ECHO KEY NAME
* SAY "GO PLEASE"
* GO GET DIGITS FROM KEYBOARD
2362 BD 27DE JSR MINTOBI
237E BD 27DB JSR DADD
2397 BD 27E1 JSR 2BEEP
23AD BD 27E7 JSR GETDIG
23A3 BD 27DB JSR PACK
23AA BD 27E1 JSR 2BEEP
23B0 BD 27F9 JSR SCAN
23B5 BD 27F0 JSR SPKEY
23C8 BD 27E1 JSR 2BEEP
23CE BD 27DB JSR GETDIG
23D5 DD 27FO JSR SPKWD
23D9 BD 27DB JSR PACK
23EE BD 27E1 JSR 2BEEP
23F0 BD 27F9 JSR GETDIG
23F3 BD 27D5 JSR PACK
23F9 BD 27DB JSR GETEM
23FE BD 27D5 JSR PACK
2404 BD 27DB JSR GETEM
2409 BD 27D5 JSR PACK
240F BD 27DB JSR GETEM
2414 BD 27D5 JSR PACK
241A BD 27DB JSR PACK

*** START OF 2BEEP SUBROUTINE ***

2455 BD 27EA JSR WAIT
245A BD 27F3 JSR LDPAUS
245F BD 27F6 JSR SPKWD
2464 BD 27F6 JSR SPKWD

*** START OF MINTOBI SUBROUTINE ***

246B BD 27DB JSR DADD

* CONVERT MMM:SS TO BIN, SECS
* TWO BYTE BINARY ADD
* SAY "GO PLEASE"
* GO GET DIGITS FROM KEYBOARD
* PACK MULTIDIGIT INFO
* SAY "GO PLEASE"
* INPUT:KEYS OUTPUT:DISP. & TALK
* ECHO KEY NAME
* SAY "GO PLEASE"
* GO GET DIGITS FROM KEYBOARD
* PACK MULTIDIGIT INFO
* SAY "GO PLEASE"
* GO GET DIGITS FROM KEYBOARD
* PACK MULTIDIGIT INFO
* START OF 2BEEP SUBROUTINE
* WAIT TILL SPEECH-BUF EMPTY
* LOAD PAUSE INTO SPEECH BUF.
* LOAD WORD INTO SPEECH BUF.
* LOAD WORD INTO SPEECH BUF.

* START OF MINTOBI SUBROUTINE
* TWO BYTE BINARY ADD
JSR DADD
JSR DADD

**** START OF SAYNUM SUBROUTINE ****

JSR SPKWD
JSR SPKWD
JSR SPKWD
JSR SPKWD
JSR SPKWD
JSR SPKWD

**** START OF SCAN SUBROUTINE ****

**** START OF CHECK SPEECH ROUTINE ****

**** START OF CHECK TIME ROUTINE ****

**** START OF KEY DEBOUNCE ROUTINE ****

JMP CONTINUE

**** START OF SPKWD AND LDPAUS SUBROUTINES ****

**** START OF GETEM SUBROUTINE ****

JSR 2BEEP
JSR GETDIG

**** START OF SPKEY SUBROUTINE ****

JSR SPKWD

**** START OF DRILL SUBROUTINE ****

JSR SPKEY
JSR SPKWD
JSR SCAN
JSR SPKEY
JSR WAIT
JSR SPKWD
JSR LDPAUS

**** START OF WAIT SUBROUTINE ****

JSR SCAN

**** START OF GETDIG SUBROUTINE ****
**** START OF BLANKIT SUBROUTINE ****
**** START OF UNPACK SUBROUTINE ****
**** START OF SAYDISP SUBROUTINE ****

275A BD 27F6    JSR SPKWD    * LOAD WORD INTO SPEECH BUF.
2766 BD 27F6    JSR SPKWD    * LOAD WORD INTO SPEECH BUF.
2772 BD 27F3    JSR LDPAUS    * LOAD PAUSE INTO SPEECH BUF.
2786 BD 27CF    JSR SAYNUM    * PRONOUNCE PACKED NUM AT (X)
278C BD 27F6    JSR SPKWD    * LOAD WORD INTO SPEECH BUF.
2792 BD 27F3    JSR LDPAUS    * LOAD PAUSE INTO SPEECH BUF.

**** START OF LINK ****
279A 7E 2ABE    JMP $2ABE    * TO TEACH
279D 7E 2924    JMP $2924    * TO COMPARE
27A0 7E 2CCF    JMP $2CCF    * TO TBOOK
27A3 7E 2869    JMP $2869    * TO GETANS
27A6 7E 2C40    JMP $2C40    * TO ADDEM
27A9 7E 281A    JMP $281A    * TO BCMDULT
27AC 7E 209F    JMP $209F    * TO RANDOM
27B1 7E 2936    JMP $2936    * TO GENERATE
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Address</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>27B4 7E 22C7</td>
<td>JMP</td>
<td>$22C7</td>
<td>* TO BCDADD</td>
</tr>
<tr>
<td>27B7 7E 22D6</td>
<td>JMP</td>
<td>$22D6</td>
<td>* TO BCDINC</td>
</tr>
<tr>
<td>27BA 7E 2C00</td>
<td>JMP</td>
<td>$2C00</td>
<td>* TO ERROR</td>
</tr>
<tr>
<td>27BD 7E 291C</td>
<td>JMP</td>
<td>$291C</td>
<td>* TO BISUB</td>
</tr>
<tr>
<td>27C0 7E 274A</td>
<td>JMP</td>
<td>$274A</td>
<td>* TO SAYDISP</td>
</tr>
<tr>
<td>27C3 7E 2710</td>
<td>JMP</td>
<td>$2710</td>
<td>* TO BLANK</td>
</tr>
<tr>
<td>27C6 7E 2670</td>
<td>JMP</td>
<td>$2F70</td>
<td>* TO DISPLAY</td>
</tr>
<tr>
<td>27C9 7E 2240</td>
<td>JMP</td>
<td>$2240</td>
<td>* TO BLANK</td>
</tr>
<tr>
<td>27CC 7E 2730</td>
<td>JMP</td>
<td>$2730</td>
<td>* TO UNPACK</td>
</tr>
<tr>
<td>27CF 7E 24A7</td>
<td>JMP</td>
<td>$24A7</td>
<td>* TO SAYNUM</td>
</tr>
<tr>
<td>27D2 7E 22F4</td>
<td>JMP</td>
<td>$22F4</td>
<td>* TO PROGRAM</td>
</tr>
<tr>
<td>27D5 7E 2612</td>
<td>JMP</td>
<td>$2612</td>
<td>* TO GETEM</td>
</tr>
<tr>
<td>27D8 7E 2CBF</td>
<td>JMP</td>
<td>$2CBF</td>
<td>* TO BADD</td>
</tr>
<tr>
<td>27DB 7E 28D8</td>
<td>JMP</td>
<td>$2FAB</td>
<td>* TO PACK</td>
</tr>
<tr>
<td>27DE 7E 2469</td>
<td>JMP</td>
<td>$2469</td>
<td>* TO MINTOBI</td>
</tr>
<tr>
<td>27E1 7E 2455</td>
<td>JMP</td>
<td>$2455</td>
<td>* TO 2BEEP</td>
</tr>
<tr>
<td>27E4 7E 2509</td>
<td>JMP</td>
<td>$2509</td>
<td>* TO CONTINUE (LOCATED IN SCAN)</td>
</tr>
<tr>
<td>27E7 7E 268D</td>
<td>JMP</td>
<td>$268D</td>
<td>* TO GETDIG</td>
</tr>
<tr>
<td>27EA 7E 2680</td>
<td>JMP</td>
<td>$2680</td>
<td>* TO WAIT</td>
</tr>
<tr>
<td>27ED 7E 2634</td>
<td>JMP</td>
<td>$2634</td>
<td>* TO DRILL</td>
</tr>
<tr>
<td>27F0 7E 2620</td>
<td>JMP</td>
<td>$2620</td>
<td>* TO SPKEY</td>
</tr>
<tr>
<td>27F3 7E 2602</td>
<td>JMP</td>
<td>$2602</td>
<td>* TO LDPAUS (LOCATED IN SPKWD)</td>
</tr>
</tbody>
</table>
**START OF BCDMULT SUBROUTINE**

```
2861 BD 27B4 JSR BCDADD
```

**ADD AIB TO LOC @X**

**START OF GETANS SUBROUTINE**

```
28A1 BD 27A6 JSR ADDEM
28AA BD 27A9 JSR BCDMULT
28E8 BD 27A6 JSR ADDEM
```

**SIGNED BCD ADDITION ROUTINE**

**SIGNED BCD MULT. ROUTINE**

**SIGNED BCD ADDITION ROUTINE**

**START OF BISUB SUBROUTINE**

```
2920 7E 27D8 JMP DADD
```

**TWO BYTE BINARY ADD**

**START OFCOMPARE SUBROUTINE**

```
2930 BD 27A6 JSR ADDEM
```

**SIGNED BCD ADDITION ROUTINE**

**START OF GENERATE SUBROUTINE**

```
294B BD 27AC JSR RANDOM
2950 BD 279D JSR COMPARE
2959 BD 279D JSR COMPARE
```

**RANDOM NUMBER GENERATOR**

**COMPARE (BCD) VALUE & X**

**COMPARE (BCD) VALUE & X**
RANDOM NUMBER GENERATOR
* COMPARE (BCD) VALUE & X
* COMPARE (BCD) VALUE & X
* GENERATE ONE OPERAND
* GENERATE ONE OPERAND
* SET C IF VALUE BETWEEN LIMITS
* SET C IF VALUE BETWEEN LIMITS

* BLANK ENTIRE DISPLAY
* SIGNED BCD ADDITION ROUTINE
* LOAD WORD INTO SPEECH BUF.
* SEPERATE NIBBLES INTO BYTES
* PRONOUNCE PACKED NUM AT (X)
* LOAD WORD INTO SPEECH BUF.
* SEPERATE NIBBLES INTO BYTES
* PRONOUNCE PACKED NUM AT (X)
* LOAD PAUSE INTO SPEECH BUF.
* WAIT TILL SPEECH-BUF EMPTY
* BLANK ENTIRE DISPLAY
* SEPERATE NIBBLES INTO BYTES
* PRONOUNCE PACKED NUM AT (X)
* LOAD WORD INTO SPEECH BUF.
* SEPERATE NIBBLES INTO BYTES
* PRONOUNCE PACKED NUM AT (X)
* LOAD WORD INTO SPEECH BUF.
* WAIT TILL SPEECH-BUF EMPTY

* PACK MULTIDIGIT INFO
* UNSIGNED BINARY SUBTRACTION
* UNSIGNED BINARY SUBTRACTION
* INCREMENT (BCD) AT X & X+1
* INCREMENT (BCD) AT X & X+1
* PRONOUNCE PROBLEM
2B25 BD 27cy JSR DISPLAY * LOAD PROB. INTO DISPLAY
2B28 BD 27C3 JSR BLANKIT * BLANK MISSING NO. IN DISP.
2B3B BD 27E7 JSR GETDIG * GO GET DIGITS FROM KEYBOARD
2B59 BD 27BA JSR ERROR * APPLY NEG. FEEDBACK
2B6B BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X & X+1
2B73 BD 27F3 JSR LDPAUS * LOAD PAUSE INTO SPEECH BUF.
2B76 BD 27C0 JSR SAYDISP * PRONOUNCE PROBLEM
2B79 BD 27C9 JSR DISPLAY * LOAD PROB. INTO DISPLAY
2B7C BD 27EA JSR WAIT * WAIT TILL SPEECH-BUF EMPTY
2B82 BD 27F9 JSR SCAN * INPUT:KEYS OUTPUT:DISP. & TALK
2B8E BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X & X+1
2B9A BD 2805 JSR RIGHT * APPLY POSITIVE FEEDBACK
2B9F BD 27B1 JSR GENERATE * GENERATE PROBLEM
2BA2 BD 27A3 JSR GETANS * FIND CORRECT ANSWER
2BDE BD 27BD JSR BISUB * UNSIGNED BINARY SUBTRACTION

*** START OF ERROR SUBROUTINE ***
2C00 BD 27EA JSR WAIT * WAIT TILL SPEECH-BUF EMPTY
2C1C BD 27F6 JSR SPKWD * LOAD WORD INTO SPEECH BUF.
2C21 BD 27F6 JSR SPKWD * LOAD WORD INTO SPEECH BUF.
2C25 BD 27C6 JSR BLANK * BLANK ENTIRE DISPLAY
2C2B BD 27F9 JSR SCAN * INPUT:KEYS OUTPUT:DISP. & TALK
2C2E BD 27C9 JSR DISPLAY * LOAD PROB. INTO DISPLAY
2C31 BD 27C3 JSR BLANKIT * BLANK MISSING NO. IN DISP.
2C37 BD 27F9 JSR SCAN * INPUT:KEYS OUTPUT:DISP. & TALK

*** START OF ADDEM SUBROUTINE ***
2C73 BD 27B4 JSR BCDADD * ADD A:B TO LOC @ X
2C94 BD 27B4 JSR BCDADD * ADD A:B TO LOC @ X
2CA9 BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X & X+1

*** START OF DADD SUBROUTINE ***

*** START OF TBL00K SUBROUTINE ***
2CF9 7E 2000 JMP $2000 * INCREMENT (BCD) AT X & X+1
2D0B BD 27B7 JSR BCDINC

*** START OF ADDTBL ***
**** START OF SUB1BL ****
**** START OF M-DTBL ****
**** START OF BLANK SUBROUTINE ****
*** START OF PACK SUBROUTINE ****
**** START OF RIGHT SUBROUTINE ****

2F86 BD 2786 JSR SPKWD * LOAD WORD INTO SPKBUF
2F8B BD 2786 JSR SPKWD * LOAD WORD INTO SPKBUF
2F90 BD 2786 JSR SPKWD * LOAD WORD INTO SPKBUF
2F93 BD 27EA JSR WAIT * WAIT UNTIL SPKBUF EMPTY

**** RESTART AND INTERRUPT VECTORS ****
*******************************************************************************
*** END OF CONTROL PROGRAMS ****
*******************************************************************************
Table III
Program Listing

This listing contains PIA addresses for D1 computer board.

Start of restart program

Start of interrupt program
**** END OF INTERUPT PROGRAM ****

**** START OF RESTART (2) PROGRAM ****

2030 CE 001D LDX $001D * INITIALIZE NEXT & LAST
2033 FF 0016 STX NEXT
2036 FF 0018 STX LAST

2039 CE 0006 LDX $0006
203C 8A FF LDA A $FF
203E A7 00 STA A 0X
2040 08 INX
2041 8C 0016 CPX $0016 * END OF DISBUF?
2044 26 FB BNE $203E

**** END OF RESTART (2) ****

**** START OF MAIN PROGRAM ****

2046 BD 27D2 JSR PROGRAM * TEACHER DEFINES PROB. SETS
2049 BD 27ED JSR DRILL * KEYBOARD TRAINING DRILL
204C BE 4A LDX SEED1 * IF SEEDS ARE EQUAL
204E 9C 62 CPX SEED2 * THEN CHANGE SEEDA
2050 26 06 BNE $2058
2052 96 48 LDA A SEED1+1
2054 88 0F EOR A $OF
2056 97 48 STA A SEED1+1
2058 96 4B LDA A MODE
205A 81 0C CMP A $0C
205C 26 03 BNE $2041 * IF PROB SET MODE THEN
205E BD 27A0 JSR TBLOOK * LOOK UP PROBLEM SET INFO
2061 01 NOP
2062 01 NOP
2063 01 NOP
2064 BD 279A JSR TEACH * TEACH MATH PROBLEMS
2067 7F 0061 CLR SAY * DISABLE REPEAT BUTTON
206A BD 27FC JSR REPORT * GIVE TOTAL RIGHT & TIME
206D BD 27C6 JSR BLANK * BLANK ENTIRE DISPLAY
159

**** END OF MAIN ****

**** START OF RANDOM SUBROUTINE ****

209F A6 00 LDA A 0,X
20A1 B4 07 AND A $07
20A3 AA 01 ORA A 1,X
20A5 26 02 BNE $20A9
20A7 63 01 COM 1,X
20A9 A6 00 LDA A 0,X
20AB 44 LSR A
20AC 44 LSR A
20AD A8 00 EOR A 0,X
20AF 44 LSR A
20B0 69 01 ROL 1,X
20B2 69 00 ROL 0,X
20B4 A6 00 LDA A 0,X
20B6 88 A5 EOR A $A5
20B8 84 03 AND A $03
20BA 36 PSH A
20BB A6 01 LDA A 1,X
20BD 88 A5 EOR A $A5
20BF CE 0000 LDX $0000
20C2 DF 75 STX VALUE
20C4 FE 27AF LDX $27AF
20C7 DF 04 STX XSAV
20C9 8D 10 BSR $20DB
20CB 32 PUL A
20CC 8D 0D BSR $20DB
20CE CE 8000 LDX $8000
20D1 9C 75 CPX VALUE
20D3 26 05 BNE $20DA
20D5 CE 8421 LDX $8421
20DB DF 75 STX VALUE

* SAY 'STOP'
* LOAD WORD INTO SPEECH BUF.
* CALL SCAN UNTIL KEY RELEASED
* INPUT KEYS OUTPUT: DISP. % TALK
* KEY=REPEAT THEN REPEAT PROBS
* ELSE REPORT

* CHECK FOR ZERO
* IN BO-B10
* REMOVE ZERO
* BO'=BO EOR B10
* DESTROY PREDICTABILITY
* CLR RUNNING TOTAL
* STARTING ADDR OF DCD TABLE
* VALUE=MINUS ZERO?
* VALUE=-421
**** START OF BCDTBL LOOK UP TABLE ****

**** END OF BCDTBL ****

**** START OF OPGEN SUBROUTINE ****
BNE $212A
BSR $212F
BRA $211F
RTS

STA ADDRESS
STX ADDRESS

JSR RANDOM
* RANDOM NUMBER GENERATOR

JSR HI-LO
* SET C IF VALUE BETWEEN LIMITS
* <499

BRA $212F
* GOTO AGAIN IF OUTSIDE LIMITS

LDA A SUM

CMP A #$08

BCE $218C

LDX VALUE

AND B #$0F

STA B VALUE
* MASK OFF HUNDREDS

LDA B VALUE+1

STA B VALUE
* BR IF MULT OF 10 REQ'D

CMP A #$06

BCE $21B6

BR IF +-9 NOT REQ'D

BNE $215C

RTS

LDA A SUM

CMP A #$04

BNE $2162

BRA $2186
* 0 TO 9

JSR HI-LO
* SET C IF VALUE BETWEEN LIMITS
* < +4
* > -4

JSR HI-LO
* HERE IF SUM=0

LDA A SUM

CMP A #$01

BNE $212F

RTS
* HERE IF SUM=1

LDA A SUM

CMP A #$02

BNE $217F

BRA $2186
* HERE IF SUM=2

BRA $218F
* HERE IF SUM=3

BRA $21B4
* STEPPING STONES

BRA $21C3
* STEPPING STONES
2188 01 NOP
2189 01 NOP
218A 01 NOP
218B 01 NOP
218C 81 0D CMP A $$0D    * SUM=0D?
218E 26 01 BNE $2191
2190 39 RTS
2191 BD 27FF JSR HI-LO  * SET C IF VALUE BETWEEN LIMITS
2194 00 99
2196 80 99
2198 25 E8 BCS $2184
219A 86 03 LDA A SUM
219C 81 0C CMP A $$0C
219E 26 23 BNE $21C3
21A0 39 RTS
21A1 81 0A CMP A $$0A
21A3 26 01 BNE $21A6
21A5 39 RTS
21A6 BD 27FF JSR HI-LO  * SET C IF VALUE BETWEEN LIMITS
21A9 00 09
21A8 80 09
21AD 25 D5 BCS $2194
21AF 86 03 LDA A SUM
21B1 81 08 CMP A $$08
21B3 27 0E BEQ $21C3
21B5 39 RTS
21B6 D6 76 LDA B VALUE+1 * MOVE ONES TO TENS PLACE
21B8 58 ASL B
21B9 60 ASL B
21BA 58 ASL B
21BB 50 ASL B
21BC D7 76 STA B VALUE+1
21BE 81 06 CMP A $$06
21C0 27 01 BEQ $21C3
21C2 39 RTS
21C3 96 75 LDA A VALUE  * ABSOLUTE VALUE SUBROUTINE
21C5 84 0F AND A $$0F
21C7 97 75 STA A VALUE
21C9 39 RTS

**** START OF HI-LO SUBROUTINE ****

21CA 80 2802 JSR LOADX  * LOAD X FROM INLINE DATA
21CD 80 279D JSR COMPARE  * COMPARE (BCD) VALUE & X
21D0 28 07 BMI $21D9
21D2 27 05 BEQ $21D9
21D4 BD 2802 JSR LOADX * LOAD X FROM INLINE DATA
21D7 OC CLC
21D8 39 RTS * HERE IF DILIN TEST FAILS
21D9 BD 2802 JSR LOADX * LOAD X FROM INLINE DATA
21DC BD 279D JSR COMPARE * COMPARE (BCD) VALUE $ X
21DF OC CLC
21E0 2A 01 BPL $21E3
21E2 39 RTS * HERE IF DOLIN FAILS TEST
21E3 OD SEC
21E4 39 RTS * HERE IF BETWEEN LIMITS

**** START OF DOWN UTILITY PROGRAM ****
220F FE 2202 LDX $2202
2212 08 INX
2213 08 INX
2214 FF 2202 STX $2202
2217 FE 2200 LDX $2200
221A A6 00 LDA A 0,X
221C 08 INX
221D FC 2202 CFX $2202
2220 27 0E BEQ $2230
2222 FF 2200 STX $2200
2225 FE 2204 LDX $2204
2228 A7 00 STA A 0,X
222A 08 INX
222B FF 2204 STX $2204
222E 20 E7 BRA $2217

**** START OF DISPLAY SUBROUTINE ****
2240 86 88 LDA A #$88
2242 97 13 STA A #$0013 * TURN ON EQUALS
2244 96 5B LDA A NUM1
2246 96 5C LDA B NUM1+1 * LOAD FIRST DISPLAY
2248 CE 0006 LDX #$0006
224B BD 27CC JSR UNPACK * SEPERATE NIBBLES INTO BYTES
224E 96 5D LDA A NUM2
2250 96 5E LDA B NUM2+1 * LOAD 2ND DISPLAY
2252 CE 000A LDX #$000A
2255 BD 27CC JSR UNPACK * SEPERATE NIBBLES INTO BYTES
2258 96 5F LDA A NUM3
225A 96 60 LDA B NUM3+1 * LOAD 3RD DISPLAY
225C CE 000E LDX #$000E
225F BD 27CC JSR UNPACK * SEPERATE NIBBLES INTO BYTES
2262 86 13 LDA A #$13
2264  C6 EF  LDA B  #EF  * LOAD OPERATION SIGN
2266  91 50  CMP A  OPSIGN
2268  27  04  BEQ $226E
226A  4A  DEC A
226B  59  ASL B
226C  20 FB  BRA $2266
226D  07  12  STA B $0012
2270  01  NOP
2271  01  NOP
2272  01  NOP
2273  CE 0007  LDX #$0007
2276  BD  08  BSR $2283
2278  CE 0009  LDX #$0009
227B  BD  06  BSR $2283
227D  CE 000F  LDX #$000F
2280  BD  01  BSR $2283
2282  39  RTS

2283  6D  00  TST 0,X  * LEADING ZERO SUPPRESS
2285  2A 08  BNE $228F
2287  63  00  COM 0,X
2289  6D  01  TST 1,X
228B  26  02  BNE $228F
228D  63  01  COM 1,X
228F  39  RTS

**** START OF UP UTILITY PROGRAM ****

229F  FE 2290  LDX $2290
22A2  09  DEX
22A3  09  DEX
22A4  FF 2290  STX $2290
22A7  FE 2292  LDX $2292
22AA  A6  00  LDA A 0,X
22AC  09  DEX
22AD  BC 2290  CPX $2290
22B0  27  0E  BEQ $22C0
22B2  FF 2292  STX $2292
22B5  FE 2294  LDX $2294
22B8  A7  00  STA A 0,X
22BA  09  DEX
22BB  FF 2294  STX $2294
22BE  20 E7  BRA $22A7

**** START OF BCDADD SUBROUTINE ****

22C7  36  PSH A
22C8  17  TBA
ADD A $1,X
DAA
STA A $1,X
TAB
PUL A
STA A 0,X
DAA
STA A 0,X
RTS

**** START OF BCDINC SUBROUTINE ****
CLR A
LDA B $01
JMP BCDADD

**** START OF LOADX SUBROUTINE ****
PSH A
PSH B
TSX
LDA A 4,X
LDA B 5,X
ADD B $02
STA A 4,X
STA B 5,X
LDX 4,X
LDX 0,X
PUL B
PUL A
RTS

**** START OF PROGRAM SUBROUTINE ****
BRA $2302
LDA A $31
STA A KEY

165
22FA 39      RTS

22FB 97 49   STA A OPTION     * PATCH TO LOAD OPTION
22FD 8E 1A   LDX SCNTIM       * AND SEED2
22FF 42      STX SEED2

2301 39      RTS

2302 CE 0001  LDX $0001      * TURN SCAN RATE TO MAX
2303 DF 46    STX RATE
2307 86 10    LDA A $010      * TURN MODE PROMPT ON
2309 97 14    STA A $0014     * SAY 'GO PLEASE'
230B BD 27E1  JSR 2BEEP
230E CE FFFE  LDX $FFFF
2311 BD 27F9  JSR SCAN       * INPUT: KEYS OUTPUT: DISP, & TALK
2314 96 00    LDA A KEY
2316 BD 27F0  JSR SPKEY      * ECHO KEY NAME
2319 96 00    LDA A KEY       * 0B=BOUNDED MODE
231B 81 0B    CMP A $008
231D 27 04    BEQ $2323      * 0C=LEVEL MODE
231F 81 0C    CMP A $00C
2321 26 E4    BNE $2307
2323 97 48    STA A MODE
2325 DE 1A    LDX SCNTIM

2327 DF 4A    STX SEED1       * LOAD RANDOM NO. SEED
2329 86 20    LDA A $020
232B 97 14    STA A $0014    * PROMPT OPTION
232D BD 27E1  JSR 2BEEP      * SAY 'GO PLEASE'
2330 CE FFFE  LDX $FFFF
2333 BD 27F9  JSR SCAN       * INPUT: KEYS OUTPUT: DISP, & TALK
2336 96 30    LDA A KEY
2338 BD 27F0  JSR SPKEY      * ECHO KEY NAME
233A 96 00    LDA A KEY       * 0D=TIMED OPTION KEY
233C 81 0D    CMP A $00D
233F 2D EB    BLT $2329
2341 81 0F    CMP A $00F
2343 2E E4    BGT $2329
2345 BD 84    BSR $22FB       * TO LOAD SEED2 & OPTION PATCH
2347 CE 0053  LDX $0053       * RESTORE NORMAL SCAN RATE
234A DF 46    STX RATE

234C 96 49    LDA A OPTION    * OPTION = POWER?
234E 81 0F    CMP A $00F
2350 27 3B    BEQ $238D
2352 86 30    LDA A $030
2354 97 14    STA A $0014    * PROMPT TIME LIMIT
2356 BD 27E1  JSR 2BEEP      * SAY 'GO PLEASE'
2359 CE FFFF  LDX $FFFF      * GO GET DIGITS FROM KEYBOARD
235B BD 27E7  JSR GETDIG      * CONVERT MMM:SS TO BIN. SECS
235F CE 004C  LDX $004C
2362 BD 27DE  JSR MINTOBI

2365 96 49    LDA A OPTION    * OPTION = SPEED?
2367 81 0E    CMP A $00E
2369 26 22    BNE $238D
TIMLIM TO SELF 50 TIMES

TO GET NUMBER OF SCANS/PROB

* TWO BYTE BINARY ADD

SET TIME LIMIT TO INFINITY

* SET TIME LIMIT TO INFINITY

TIMED?

* PROMPT NO. OF PROBLEMS

SAY 'GO PLEASE'

PROMPT NO. OF PROBLEMS

GO GET DIGITS FROM KEYBOARD

PACK MULTIDIGIT INFO

PROMPT MISSING NO.

INPUT:KEYS OUTPUT:DISP. & TALK

CHECK FOR LEGAL KEY

PROMPT OPERATION SIGN

GO PLEASE'

PROMPT LEVEL OF DIFFICULTY
**START OF 2BEEP SUBROUTINE ****

2455 BD 27E1 JSR 2BEEP * SAY '00 PLEASE'
2454 CE FFFF LDX $FFF * GO GET DIGITS FROM KEYBOARD
2453 BD 27E7 JSR GETDIG
2452 CE 0053 LDX $0053 * PACK MULTIDIGIT INFO
2451 BD 27DB JSR PACK
2450 39 RTS

23F1 86 00 LDA A $90 * PROMPT FOR HILIM1
23F3 BD 27D5 JSR GETEM * PROMPT & SCAN FOR RESPONSE
23F6 CE 0053 LDX $0053 * PACK MULTIDIGIT INFO
23F9 BD 27DB JSR GETEM
23FE BD 27D5 JSR GETEM * PROMPT FOR LOLIM1
2401 CE 0055 LDX $0055
2404 BD 27DB JSR GETEM * PROMPT & SCAN FOR RESPONSE
2407 BD 80 LDA A $90 * PACK MULTIDIGIT INFO
2409 BD 27D5 JSR GETEM
240C CE 0057 LDX $0057
240F BD 27DB JSR GETEM
2412 BD 80 LDA A $90 * PACK MULTIDIGIT INFO
2414 BD 27D5 JSR GETEM
2417 CE 0059 LDX $0059
241A BD 27DB JSR GETEM
241D 39 RTS

**** START OF MINTOBI SUBROUTINE ****

2469 CE 05 BRA $2470 * TWO BYTE BINARY ADD
2468 BD 27DB JSR DADD
2467 29 BRA $2498

2470 6F 00 CLR 0,X
2472 6F 01 CLR 1,X
2474 96 3E LDA A DIG1
2476 A7 01 STA A 1,X * SECONDS
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TST DIG2</td>
<td></td>
</tr>
<tr>
<td>BEQ $2488</td>
<td></td>
</tr>
<tr>
<td>CLR A</td>
<td></td>
</tr>
<tr>
<td>LDA B #$0A</td>
<td>* MULT DIG2 BY 10 % ADD TO VAL.</td>
</tr>
<tr>
<td>JSR DADD</td>
<td>* TWO BYTE BINARY ADD</td>
</tr>
<tr>
<td>DEC DIG3</td>
<td></td>
</tr>
<tr>
<td>BRA $2468</td>
<td></td>
</tr>
<tr>
<td>TST DIG4</td>
<td></td>
</tr>
<tr>
<td>BEQ $24A6</td>
<td></td>
</tr>
<tr>
<td>CLR A</td>
<td></td>
</tr>
<tr>
<td>LDA B #$02</td>
<td></td>
</tr>
<tr>
<td>JSR DADD</td>
<td></td>
</tr>
<tr>
<td>DEC DIG4</td>
<td></td>
</tr>
<tr>
<td>BRA $2468</td>
<td></td>
</tr>
<tr>
<td>TST DIG5</td>
<td></td>
</tr>
<tr>
<td>BEQ $24D9</td>
<td></td>
</tr>
<tr>
<td>CLR A</td>
<td></td>
</tr>
<tr>
<td>LDA B #$02</td>
<td></td>
</tr>
<tr>
<td>JSR DADD</td>
<td></td>
</tr>
<tr>
<td>DEC DIG5</td>
<td></td>
</tr>
<tr>
<td>BRA $24D9</td>
<td></td>
</tr>
<tr>
<td>TST DIG6</td>
<td></td>
</tr>
<tr>
<td>BEQ $24D9</td>
<td></td>
</tr>
<tr>
<td>LDA A 0,X</td>
<td></td>
</tr>
<tr>
<td>LDA B 1,X</td>
<td></td>
</tr>
<tr>
<td>TST A</td>
<td></td>
</tr>
<tr>
<td>BRA $24F0</td>
<td></td>
</tr>
<tr>
<td>PSH A</td>
<td></td>
</tr>
<tr>
<td>AND A #$0F</td>
<td>* HERE IF NO IS MINUS</td>
</tr>
<tr>
<td>LDA A #$1F</td>
<td>* SAY 'MINUS'</td>
</tr>
<tr>
<td>JSR SPKWD</td>
<td>* LOAD WORD INTO SPEECH BUF.</td>
</tr>
<tr>
<td>PUL A</td>
<td></td>
</tr>
<tr>
<td>AND A #$0F</td>
<td></td>
</tr>
<tr>
<td>LDA A #$1F</td>
<td></td>
</tr>
<tr>
<td>JSR SPKWD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*** START OF SAYNUM SUBROUTINE ****

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA A 0,X</td>
<td></td>
</tr>
<tr>
<td>LDA B 1,X</td>
<td></td>
</tr>
<tr>
<td>TST A</td>
<td></td>
</tr>
<tr>
<td>BRA $24F0</td>
<td></td>
</tr>
<tr>
<td>PSH A</td>
<td></td>
</tr>
<tr>
<td>AND A #$0F</td>
<td>* HERE IF NO IS MINUS</td>
</tr>
<tr>
<td>LDA A #$1F</td>
<td>* SAY 'MINUS'</td>
</tr>
<tr>
<td>JSR SPKWD</td>
<td>* LOAD WORD INTO SPEECH BUF.</td>
</tr>
<tr>
<td>PUL A</td>
<td></td>
</tr>
<tr>
<td>AND A #$0F</td>
<td></td>
</tr>
<tr>
<td>LDA A #$1F</td>
<td></td>
</tr>
<tr>
<td>JSR SPKWD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR SPKWD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR LDPAUS</td>
<td></td>
</tr>
<tr>
<td>PUL B</td>
<td></td>
</tr>
<tr>
<td>CMP B #$10</td>
<td>* BRANCH IF TEENS</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>BLT $24D9</td>
<td></td>
</tr>
<tr>
<td>TBA</td>
<td></td>
</tr>
</tbody>
</table>
START OF SCAN SUBROUTINE

**** START OF SCAN SUBROUTINE ****

24CE 44 LSR A
24CF 44 LSR A
24D0 44 LSR A
24D1 44 LSR A
24D2 88 12 ADD A $112 * ADJUST TO SAY TENS DIGIT
24D4 BD 27F6 JSR SPKWD * LOAD WORD INTO SPEECH BUF.
24D7 20 0D BRA $24E6
24D9 C5 10 BIT B $110 * HERE IF TEENS
24DB 27 09 BEQ $24E6
24DF 17 TBA
24DE 84 0F AND A $10F
24E0 88 0A ADD A $10A * SAY (ONES DIGIT + 10)
24E2 BD 27F6 JSR SPKWD * LOAD WORD INTO SPEECH BUF.
24E5 39 RTS
24E6 C4 0F AND B $10F
24E8 26 01 BNE $24EB * RTS IF ONES DIGIT = 0
24EA 39 RTS
24EB 17 TBA
24EC BD 27F6 JSR SPKWD * LOAD WORD INTO SPEECH BUF.
24EF 39 RTS
24F0 26 09 BNE $24FB
24F2 5D TST B
24F3 26 06 BNE $24FB
24F5 86 00 LDA A $000 * HERE IF NUM = 0
24F7 BD 27F6 JSR SPKWD * LOAD WORD INTO SPEECH BUF.
24FA 39 RTS
24FB 4D TST A
24FC 2A 87 SPL $24E5
24FE 20 AE BRA $24AE

**** START OF SCAN SUBROUTINE ****

2500 FF 001A STX SCNTIM * SAVE NUM OF SCANS TO DO
2503 7F 0000 CLR KEY * INITIALIZE
2506 7F 0001 CLR BF
2509 7F 0002 CLR DADDR
250C 7F 0003 CLR SUM
250F CE 0006 LDX $0006 * $0006 IS START OF DISBUF
2512 FF 0004 STX XSAV
2515 FE 0004 LDX XSAV * OUTPUT DATA AND DEV ADDR
2518 A6 00 LDA A $0X
251A 84 F0 AND A $F0
251C 8A 002 DRA A DADDR
251F B7 8008 STA A PORT-A
171

2522 BD 2D BSR $2551 * STROBE DECODER & DELAY
2524 01 NOP
2525 01 NOP
2526 01 NOP
2527 84 03 AND A $03 * MASK OFF ALL BUT KEY BITS
2529 36 PSH A
252A 44 LSR A
252B 86 0003 ADC A SUM * SUM CONTAINS $ OF KEYS DOWN
252E B7 0003 STA A SUM
2531 32 PUL A
2532 4D TST A
2533 27 0D BEQ $2542 * BR IF NO KEY THIS ADDR
2535 44 LSR A
2536 7B 0002 ASL DA0DR
2539 B6 0002 ORA A DA0DR * GENERATE CODE FOR KEY
253C B7 0000 STA A KEY
253F 77 0002 ASR DA0DR
2542 7C 0002 INC DA0DR * NEXT DEV. ADDRESS
2545 7C 0005 INC XSAV+1 * NEXT BUFFER ADDRESS
2548 B6 0002 LDA A DA0DR
254B 81 10 CMP A $10 * DA0DR BEYOND RANGE?
254D 26 C6 BNE $2515
2551 B6 34 LDA A $34
2553 B7 8009 STA A CONT-A * SET CA2 LOW TO STROBE DEC.
2556 FE 0046 LDX RATE
2559 FE 0046 DEX * DELAY LOOP TO SET SCAN RATE
255A B6 FD BNE $2559 * TO 50 SCANS/SEC
255C B6 800A LDA A PORT-B * BRING IN KEY BITS
255F 36 PSH A
2560 B6 3C LDA A $3C
2562 B7 8009 STA A CONT-A * SET CA2 HIGH
2565 32 PUL A
2566 39 RTS

**** START OF CHECK SPEECH ROUTINE ****

2567 01 NOP
2568 06 CLI
2569 B6 08 LDA A $08 * BRANCH IF START = 0
256B B5 800B BIT A CONT-B
256E 27 4A BEQ $25BA
2570 FE 0016 LDX NEXT
2573 BC 0018 CFX LAST
2576 27 42 BEQ $25BA
2579 7D 001C TST PAUSE
257B 26 2B BNE $2548
257D B6 800A LDA A PORT-B
AND A #$FC * MASK OFF KEY INPUTS
CMX A 0,X
BNE $2590 * BR IF NEXT WORD NOT AT PORTB
LDA #$35
STA CONT-B * START=0 (BEGIN TALKING)
INC NEXT * POINT TO NEXT WORD

LDA #$03 * IF 2 LSB'S OF NEXT WORD = 0
LDX NEXT * THEN WORD IS VALID
LDA A 0,X
STA PAUSE * LOAD NEXT "WORD" INTO PAUSE
INC NEXT * FLAG INDICATING PAUSE

BRA $25BA

LDA #$01 * 20MS PAUSE BETWEEN WORDS
STA PAUSE
LDA A 0,X
STA PORT-B

LDA FE 0016 LDX NEXT * OUTPUT NEXT WORD TO PORT-B

LDA FE 0016 LDX NEXT * CORRECT NEXT IF IT IS

CPX #$003D CPX #$0000D * BEYOND SPEECH BUFFER

LDA 25C5 BNE #25C5

LDX #2001D STX NEXT

STX SCNTIM

DEC PAUSE

BRA $25BA

LDA #$01
Nop

LDA 01
Nop

LDA 01
Nop

**** START OF CHECK TIME ROUTINE ****

LDX SCNTIM

CPX #$FFF

BNE #25D6

DEX

BNE #25D6

RTS

STX SCNTIM

Nop

Nop

Nop

Nop
**** START OF KEY DEBOUNCE ROUTINE ****

* SUM=0 BF=0 ACTION=NONE
* SUM=1 BF=0 ACTION= BF=1 WAIT FOR RELEASE
* SUM=1 BF=1 ACTION= BF=2 INPUT BOUNCE
* SUM=0 BF=2 ACTION= RTS KEY RELEASED !
* SUM>1 BF=X ACTION= BF=0 TOO MANY KEYS

25DB B6 0003 LDA A SUM
25DE 81 01 CMP A #$01
25E0 2E 0A BGT $25EC
25E2 27 00 BEQ $25F1
25E4 B6 0001 LDA A BF
25E7 B1 02 CMP A #$02
25E9 26 01 BNE $25EC
25EB 39 RTS

25EC 7F 0001 CLR BF
25EF 20 0A BRA $25FB

25F1 B6 0001 LDA A BF
25F4 B1 02 CMP A #$02
25F6 27 03 BEQ $25FB
25FA 7C 0001 INC BF
25FB 7E 27E4 JMP CONTINUE * MAKE ANOTHER SCAN

**** START OF SPKWD AND LDPAUS SUBROUTINES ****

2600 48 ASL A * ENTER HERE FOR SPKWD
2601 48 ASL A * SHIFTS WORD

2602 BE 18 LDX LAST * ENTER HERE FOR LDPAUS
2604 A7 00 STA A O+x
2606 08 INX
2607 8C 003D CPX #$003D * ADJ LAST IF OUTSIDE
260A 26 03 BNE $260F * OF SPEECH BUFFER
260C CE 001D LDX #$001D
260F DF 18 STX LAST
2611 39 RTS

**** START OF GETEM SUBROUTINE ****

2612 97 14 STA A $0014 * STORE A IN PROMPT BUF
**** START OF SPKEY SUBROUTINE ****

2620 B1 0A  CMP A $0A  * KEY = MINUS
2622 2E 08  BGT $262C  * KEY HAS NO NAME
2624 26 02  BNE $262B  * KEY NAME=KEY VALUE
2626 B6 1F  LDA A $1F  * 'MINUS'
2628 BD 27F6  JSR SPKWD  * LOAD WORD INTO SPEECH BUF.
262A 39  RTS
262C B6 3E  LDA A $3E  * 'HI-TONE'
262E 20 FB  BRA $2628

**** START OF DRILL SUBROUTINE ****

2634 B6 00  LDA A $00  * PROMPT KEYBOARD
2636 97 14  STA A $0014
2638 5F  CLR B
263A BD 27F0  TRA SPKEY  * SAY NEXT KEY
263C B6 2F  LDA A $2F  * 'PLEASE'
263E BD 27F6  JSR SPKWD  * LOAD WORD INTO SPEECH BUF.
2642 CE 00FA  LDX $00FA  * SCAN FOR 5 SEC. (250 SCANS)
2644 BD 27F9  JSR SCAN  * INPUT:KEYS  OUTPUT:DISP. & TALK
2646 7D 0001  TST BF  * BF=0 IMPLIES TIMED OUT
2648 27 EC  BEQ $2639
264A 20 1B  BRA $266A
264C 36  PSH A
2650 BD 27F0  JSR SPKEY  * ECHO KEY NAME
2652 BD 27EA  JSR WAIT  * WAIT TILL SPEECH-BUF EMPTY
2654 32  PUL A
2656 11  CBA
2658 26 E8  BNE $2642  * KEY CORRECT?
265A B6 3E  LDA A $3E  * 'HI-TONE'
265C BD 27F6  JSR SPKWD  * LOAD WORD INTO SPEECH BUF.
265E B6 25  LDA A $25  * 37 SCANS
2660 BD 27F3  JSR LDPAUS  * LOAD PAUSE INTO SPEECH BUF.
2662 5C  INC B
2665 C1 0B  CMP B $0B  * LAST KEY?
2667 26 DO  BNE $2639

* SAY "GO PLEASE"
* GO GET DIGITS FROM KEYBOARD

SAY "GO PLEASE"
GET DIGITS FROM KEYBOARD

KEY NAME=KEY VALUE
'MINUS'
LOAD WORD INTO SPEECH BUF.
'HI-TONE'
PROMPT KEYBOARD
SAY NEXT KEY
ECHO KEY NAME
'PLEASE'
LOAD WORD INTO SPEECH BUF.
SCAN FOR 5 SEC. (250 SCANS)
INPUT:KEYS  OUTPUT:DISP. & TALK
BF=0 IMPLIES TIMED OUT
KEY CORRECT?
'HI-TONE'
LOAD WORD INTO SPEECH BUF.
37 SCANS
LOAD PAUSE INTO SPEECH BUF.
LAST KEY?
START OF WAIT SUBROUTINE ****

2680 CE 000A LDX #$000A
2683 BD 27F9 JSR SCAN
2686 DE 16 LDX NEXT
2688 9C 18 CPX LAST
268A 26 F4 BNE $2680
268C 39 RTS

**** START OF GETDIG SUBROUTINE ****

268B DF 44 STX TIMER
268E CE 003D LDX #$003D
2692 6F 00 CLR O,X
2694 08 INX
2695 8C 0044 CPX $0044
2698 26 F8 BNE $2680
269A DE 44 LDX TIMER
269C BD 27F9 JSR SCAN
269F 7D 003D TST SGNDIG
26A2 26 04 BNE $26A8
26A4 DE 1A LDX SCNTIM
26A6 DF 44 STX TIMER
26A8 20 14 BRA $26B4
26A9 96 00 LDA A KEY
26AC 20 36 BRA $26E4
26AE 01 NOP
26AF 81 0A CMP A #$0A
26B1 26 12 BNE $26C5
26B3 86 80 LDA A #$80
26B5 94 3D ORA SGNDIG
26B7 97 3D STA SGNDIG
26B9 FE 0044 LDX TIMER
26BC 20 DE BRA $269C
26BE 96 01 LDA A BF
26C0 81 02 CMP A #$02
26C2 27 E6 BEQ $26AA
26C4 39 RTS
26C5 2F 03  BLE  $26CA  * KEY NOT VALID  
26C7 20 22  BRA  $26EB  
26C9 01  NOP  

26CA 7C 003D  INC  SGNDIG  
26CD CE 003E  LDX  #003E  * PUSH DOWN DIGITS  
26D0 E6 00  LDA B  0,X  * SO LAST ENTRY AT DIG1  
26D2 A7 00  STA A  0,X  
26D4 17  TBA  
26D5 08  INX  
26D6 8C 0044  CFX  #0044  
26D9 26 F5  BNE  $26D0  
26DB 96 3D  LDA A  SGNDIG  
26D8 B4 7F  AND A  #7F  
26DF 81 06  CMP A  #06  * IF NO DIGITS > OR = 6 RTS  
26E1 2D D6  BLT  $26B9  
26E3 39  RTS  

26E4 BD 27F0  JSR  SPKEY  * ECHO KEY NAMES  
26E7 96 00  LDA A  KEY  * THIS PATCH COMPANATES FOR  
26E9 20 C4  BRA  $26AF  * SPKEY DESTROYING A-REG  
26EB 81 17  CMP A  #17  * KEY=REPEAT?  
26ED 26 08  BNE  $26F7  
26EF BD 27CO  JSR  SAYDISP  * PRONOUNCE PROBLEM  
26F2 BD 27EA  JSR  WAIT  * WAIT TILL SPEECH-BUF EMPTY  
26F5 20 08  BRA  $26FF  
26F7 81 16  CMP A  #16  * KEY=ENTER?  
26F9 28 01  BNE  $26FC  * KEY=ENTER THEN RTS  
26FB 39  RTS  
26FC 9D 27E1  JSR  28EFP  * SAY 'GO PLEASE'  
26FF DE 44  LDX  TIMER  
2701 20 8A  BRA  $26BD  * START OVER  

**** START OF BLANKIT SUBROUTINE ****  

2710 96 52  LDA A  MISSING#1  
2712 CE 000A  LDX  #000A  * THIS ROUTINE BLANKS  
2715 B1 02  CMP A  #002  * THE MISSING NUMBER IN  
2717 2D 0D  BLT  $2726  * THE DISPLAY BY PUTTING  
2719 2E 10  BGT  $2728  * #$ IN BUFFER  
271B 86 FF  LDA A  #$FF  
271D A7 00  STA A  0,X  
271F A7 01  STA A  1,X  
2721 A7 02  STA A  2,X  
2723 A7 03  STA A  3,X  
2725 39  RTS
**** START OF UNPACK SUBROUTINE ****

2730 37  PSH B
2731 36  PSH A
2732 84 F0  AND A $F0
2734 A7 00  STA A 0,X
2736 32  PUL A
2737 48  ASL A
2738 48  ASL A
2739 48  ASL A
273A 48  ASL A
273B A7 01  STA A 1,X
273D C4 F0  AND B $F0
273E E7 02  STA B 2,X
2741 32  PUL A
2742 48  ASL A
2743 48  ASL A
2744 48  ASL A
2745 48  ASL A
2746 A7 03  STA A 3,X
2748 39  RTS

**** START OF SAYDISP SUBROUTINE ****

274A 7D 0061  TST SAY
274D 27 40  BEQ $278F
274F C6 01  LDA B $F1
2751 CE 005B  LDX $F005B
2754 8D 20  BSR $2776
2756 96 50  LDA A OPSIGN
2758 8B 0E  ADD A $0E
275A BD 27F6  JSR SPKWD
275D C6 02  LDA B $02
275F CE 005D  LDX $F005D
2762 8D 12  BSR $2776
LDA A #$22  * 'EQUALS'  
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.  
LDA B #$03  * 3RD NUMBER  
LDX #$005F  
BSR $2776  
LDA A #$19  * 1/2 SEC PAUSE  
JSR LDPAUS  * LOAD PAUSE INTO SPEECH BUF.  

LDA A #$15  * 2/5 SEC PAUSE BETWEEN NUMS.  
BRA $2790  * LOAD PAUSE  
LDA A SAY  
CMP A #$02  
BEQ $278A  * IF = SAY BEEP  
JSR SAYNUM  * PRONOUNCE PACKED NUM AT (X)  
TS  
LDA A #$3D  * 'LO-TONE'  
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.  

STX XSAV  * LOAD PAUSE INTO SPEECH BUF.  
JSR LDPAUS  
LDX XSAV  
BRA $2778  

**** START OF LINK ****

JMP $2A8E  * TO TEACH  
JMP $2924  * TO COMPARE  
JMP $2CCF  * TO TBL00K  
JMP $2869  * TO GETANS  
JMP $2C40  * TO ADDEM  
JMP $281A  * TO BCDMULT  
JMP $209F  * TO RANDOM  
JMP $209F  * STARTING ADDR OF BCDTBL  
JMP $2936  * TO GENERATE  
JMP $22C7  * TO BCDA00D  
JMP $22D6  * TO BCDINC  
JMP $2C00  * TO ERROR  
JMP $291C  * TO BISUB  
JMP $274A  * TO SAYDISP  
JMP $2710  * TO BLANKIT  
JMP $2F70  * TO BLANK  
JMP $2240  * TO DISPLAY  
JMP $2730  * TO UNPACK  
JMP $24A7  * TO SAYNUM
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>27D2</td>
<td>7E 22F4</td>
<td>JMP $22F4</td>
<td>* TO PROGRAM</td>
</tr>
<tr>
<td>27D5</td>
<td>7E 2612</td>
<td>JMP $2612</td>
<td>* TO GETEM</td>
</tr>
<tr>
<td>27D6</td>
<td>7E 2C8F</td>
<td>JMP $2C8F</td>
<td>* TO DADD</td>
</tr>
<tr>
<td>27D8</td>
<td>7E 2FAB</td>
<td>JMP $2FAB</td>
<td>* TO PACK</td>
</tr>
<tr>
<td>27D9</td>
<td>7E 2469</td>
<td>JMP $2469</td>
<td>* TO MINTOBI</td>
</tr>
<tr>
<td>27DE</td>
<td>7E 2455</td>
<td>JMP $2455</td>
<td>* TO 2BEEP</td>
</tr>
<tr>
<td>27D4</td>
<td>7E 2509</td>
<td>JMP $2509</td>
<td>* TO CONTINUE (LOCATED IN SCAN)</td>
</tr>
<tr>
<td>27D7</td>
<td>7E 268D</td>
<td>JMP $268D</td>
<td>* TO GETDIG</td>
</tr>
<tr>
<td>27D5</td>
<td>7E 2680</td>
<td>JMP $2680</td>
<td>* TO WAIT</td>
</tr>
<tr>
<td>27D6</td>
<td>7E 2634</td>
<td>JMP $2634</td>
<td>* TO DRILL</td>
</tr>
<tr>
<td>27D8</td>
<td>7E 2620</td>
<td>JMP $2620</td>
<td>* TO SPKEY</td>
</tr>
<tr>
<td>27F3</td>
<td>7E 2602</td>
<td>JMP $2602</td>
<td>* TO LDPAUS (LOCATED IN SPKWD)</td>
</tr>
<tr>
<td>27F4</td>
<td>7E 2600</td>
<td>JMP $2600</td>
<td>* TO SPKWD (SEE LDPAUS)</td>
</tr>
<tr>
<td>27F9</td>
<td>7E 2500</td>
<td>JMP $2500</td>
<td>* TO SCAN (SEE CONTINUE)</td>
</tr>
<tr>
<td>27F6</td>
<td>7E 2A43</td>
<td>JMP $2A43</td>
<td>* TO REPORT</td>
</tr>
<tr>
<td>27F7</td>
<td>7E 21CA</td>
<td>JMP $21CA</td>
<td>* TO HI-LO</td>
</tr>
<tr>
<td>27F7</td>
<td>7E 22BC</td>
<td>JMP $22BC</td>
<td>* TO LOADX</td>
</tr>
<tr>
<td>27D8</td>
<td>7E 2F80</td>
<td>JMP $2F80</td>
<td>* TO RIGHT</td>
</tr>
<tr>
<td>27F8</td>
<td>7E 2F11</td>
<td>JMP $2F11</td>
<td>* TO OPGEN</td>
</tr>
</tbody>
</table>

**ADDTRBL STARTING ADDRESS**

**SUBTRBL STARTING ADDRESS**

**MULT-DIV TBL STARTING ADDRESS**

**** START OF BCDMULT SUBROUTINE ****

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>281A</td>
<td>DE 5B</td>
<td>LDX NUM1</td>
<td>* STORE nums IN TEMP VARS.</td>
</tr>
<tr>
<td>281C</td>
<td>DF 77</td>
<td>STX UNE</td>
<td></td>
</tr>
<tr>
<td>281E</td>
<td>DE 5B</td>
<td>LDX NUM2</td>
<td></td>
</tr>
<tr>
<td>2822</td>
<td>8D 0F</td>
<td>STX DEUX</td>
<td>* UNSIGNED BCD MULT.</td>
</tr>
<tr>
<td>2824</td>
<td>96 5B</td>
<td>LDX DEUXH</td>
<td></td>
</tr>
<tr>
<td>2826</td>
<td>9B 5D</td>
<td>EOR A NUM1</td>
<td>* SIGN= SGN1 EXOR SGN2</td>
</tr>
<tr>
<td>2828</td>
<td>84 80</td>
<td>AND A #$80</td>
<td></td>
</tr>
<tr>
<td>282A</td>
<td>9A 7B</td>
<td>ORA A TROIX</td>
<td></td>
</tr>
<tr>
<td>282C</td>
<td>97 7B</td>
<td>STA A TROIX</td>
<td></td>
</tr>
<tr>
<td>282E</td>
<td>DE 7B</td>
<td>LDX TROIX</td>
<td></td>
</tr>
<tr>
<td>2830</td>
<td>DF 5F</td>
<td>STX NUM3</td>
<td></td>
</tr>
<tr>
<td>2832</td>
<td>39</td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td>2833</td>
<td>7F 0077</td>
<td>CLR UNE</td>
<td>* GET RID OF SIGNS</td>
</tr>
<tr>
<td>2836</td>
<td>7F 007B</td>
<td>CLR TROIX</td>
<td></td>
</tr>
<tr>
<td>2839</td>
<td>7F 007C</td>
<td>CLR TROIX+1</td>
<td>* CLR RUNNING TOTAL</td>
</tr>
<tr>
<td>283C</td>
<td>96 7A</td>
<td>LDA A DEUX+1</td>
<td></td>
</tr>
<tr>
<td>283E</td>
<td>84 0F</td>
<td>AND A #$0F</td>
<td></td>
</tr>
<tr>
<td>2840</td>
<td>8D 14</td>
<td>BSR #$2856</td>
<td></td>
</tr>
<tr>
<td>2842</td>
<td>CA 04</td>
<td>LDA B #$04</td>
<td></td>
</tr>
<tr>
<td>2844</td>
<td>96 7A</td>
<td>LDA A DEUX+1</td>
<td></td>
</tr>
<tr>
<td>2846</td>
<td>5D</td>
<td>TST B</td>
<td></td>
</tr>
<tr>
<td>2847 27 0A</td>
<td>BEQ $2853</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2849 78 007B</td>
<td>ASL UNE+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>284C 79 0077</td>
<td>ROL UNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>284F 44</td>
<td>LSR A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2850 5A</td>
<td>DEC B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2851 20 F3</td>
<td>BRA $2846</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2853 8D 01</td>
<td>BSR $2856</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2855 39</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2856 4D</td>
<td>TST A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2857 27 0F</td>
<td>BEQ $2860</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2859 36</td>
<td>PSH A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>285A 96 77</td>
<td>LDA A UNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>285C D6 78</td>
<td>LDA B UNE+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>285D CE 007B</td>
<td>LDX #$007B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2861 BD 27B4</td>
<td>JSR $2868</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2864 32</td>
<td>PUL A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2865 4A</td>
<td>DEC A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2866 20 EE</td>
<td>BRA $2856</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2868 39</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**** START OF GETANS SUBROUTINE ****

| 2869 BD 20 | BSR $2869 |
| 286B CE 005B | LDX #$005B |
| 286D BD 0B | BSR $2878 |
| 2870 CE 005D | LDX #$005C |
| 2873 BD 06 | BSR $2878 |
| 2875 CE 005F | LDX #$005F |
| 2878 BD 01 | BSR $287B |
| 287A 39 | RTS |
| 287B A6 00 | LDA A 0,X |
| 287D B1 80 | CMP A #$80 |
| 287F 26 09 | BNE $288A |
| 2881 A6 01 | LDA A 1,X |
| 2883 4D | TST A |
| 2884 26 04 | BNE $288A |
| 2886 6F 00 | CLR 0,X |
| 2888 6F 01 | CLR 1,X |
| 288A 39 | RTS |
| 288B DE 5B | LDX NUM1 |
| 2890 DF 77 | STX UNE |
| 2898 DE 5D | LDX NUM2 |
| 2891 DF 79 | STX DEUX |
| 2893 96 50 | LDA A OPSIGN |
| 2895 B1 11 | CMP A #$11 |
| 2897 2E 11 | BGT $28AA |
| 2899 26 06 | BNE $28A1 |

* BEQ = * if equal
* ASL = Arithmetic shift left
* LSR = Logical shift right
* STA = Store accumulator
* STX = Store X-register
* LDA = Load accumulator
* LDX = Load X-register
* STA = Store accumulator
* STX = Store X-register
* LDA = Load accumulator
* LDX = Load X-register
SIGNED BCD ADDITION ROUTINE

JSR ADDEM

SIGNED BCD MULT. ROUTINE

JSR BCDMULT

OPSIGN = MULT.?

BEQ $28A5

FOF, DIVISION OF NUMBERS

LDX NUM2

STX NUM3

AFTER DOING MULT.

LDX NUM1

STX NUM2

DIVIDE BY ZERO?

LDX NUM2

STX NUM1

CHANGE PROB TO 0/1=0

LDX $40001

STX NUM2

HERE IF SUBTRACTION

LDA MODE

BR IF MODE NOT EQUAL LEVEL

CMP A $40C

BNE $28B5

PROTOTYPE

LDA A $40

AND A $40

BEQ $28F8

BR IF BIT6 NOT SET (T-BIT)

JSR ADDEM

SIGNED BCD ADDITION ROUTINE

IF HERE: DO SUBTRACTION BY

LDX NUM3

DOING ADDITION THEN MOVE SUM

STX NUM2

TO NUM1, NUM1 TO NUM2, AND

LDX TROIX

NUM2 TO NUM3 (NOT IN THAT ORDER)

RTS

LDA A DEUX

EOR A $880
**** START OF BISUB SUBROUTINE ****

291C 40  NEG A
291D 50  NEG B
291E 82 00  SBC A  $00
2920 7E 27D8  JMP   DADD  * TWO BYTE BINARY ADD

**** START OF COMPARE SUBROUTINE ****

2924 DF 79  STX  DEUX  * VALUE-X=TROIX (BCD)
2926 DE 75  LDX  VALUE
2928 DF 77  STX  UNE
292A 96 79  LDA  A  DEUX
292C 88 80  EOR A  $80
292E 97 79  STA  A  DEUX
2930 BD 27A6  JSR   ADDEM  * SIGNED BCD ADDITION ROUTINE
2933 DE 78  LDX  TROIX
2935 39  RTS

**** START OF GENERATE SUBROUTINE ****

2936 20 0A  BRA  $2942  * BRANCH AROUND PATCH
2938 7D 0055  TST  LOLIM1/PROBTYP
2938 2B 01  BMI  $293E  * IF MSB=1 CHECK THAT
293D 39  RTS  * ADS VAL OF OPERANDS ARE
293E DE 5B  LDX  NUM1  * LESS THAN OR EQUAL TO 30
2940 20 66  BRA  $29A8
2942 96 48  LDA  A  MODE
2944 81 0C  CMP A  $00C
2946 27 38  BEQ  $29B0  * BR IF MODE=LEVELS
2948 CE 007D  LDX  $1007D  * ADDR OF SEED
2948 BD 27AC  JSR  RANDOM  * RANDOM NUMBER GENERATOR
294E DE 53  LDX  LEVEL/HILIM1  * CMP, RANDOM NUM TO LIMIT
2950 BD 279D  JSR  COMPARE  * COMPARE (BCD) VALUE & X
2953 27 02  BEQ  $2957  * BR IF IN LIMITS
2955 2A F1  BPL  $2948  * BR IF OUT OF LIMITS
2957 DE 55 LDX LOLIM1/PROBTYP
2959 BD 279D JSR COMPARE * COMPARE (BCD) VALUE & X
295C 2B EA BMI $294B * BR IF BELOW LIMITS
295E DE 75 LDX VALUE
2960 DF 58 STX NUM1 * MOVE RANDOM NUM
2962 CE 007F LDX #007F * ADDR OF SEEDEB
2965 BD 27AC JSR RANDOM * RANDOM NUMBER GENERATOR
2968 DE 57 LDX HILIM2
296A BD 279D JSR COMPARE * COMPARE (BCD) VALUE & X
296D 27 02 BEq $2971
296F 2A F1 BPL $2962
2971 DE 59 LDX LOLIM2
2973 BD 279D JSR COMPARE * COMPARE (BCD) VALUE & X
2976 2B EA BMI $2962
2978 DE 75 LDX VALUE
297A DF 5D STX NUM2 * STORE 2ND OPERAND
297C 01 NOP
297D 01 NOP
297E 01 NOP
297F 39 RTS * END OF ROUTINE TO
2980 96 56 LDA A (LOLIM/PROBTYP)+1
2982 36 PSH A * STORE INFO ON 2ND OPERAND
2983 84 F0 AND A #F0 * MASK OFF ALL BUT 1ST INFO
2985 44 LSR A
2986 44 LSR A
2987 44 LSR A
2988 44 LSR A
2989 CE 007D LDX #007D * GENERATE NUM 1 FROM SEEDEA
298C BD 2808 JSR OPGEN * GENERATE ONE OPERAND
298F DE 75 LDX VALUE
2991 DF 58 STX NUM1
2993 32 PUL A * RECOVER INFO ON 2ND OPR.
2994 84 0F AND A #0F
2996 CE 007F LDX #007F * GENERATE NUM2 FROM SEEDEB
2999 BD 2808 JSR OPGEN
299C DE 75 LDX VALUE
299E DF 3D STX NUM2
29A0 96 50 LDA A OPSIGN
29A2 81 11 CMP A #11
29A4 2F 1B BLE $29C1 * BR IF ADD OR SUBTRACT
29A6 20 90 BRA $293B * BR TO PATCH
29A8 DF 75 STX VALUE * TEST IF OPR.'S WITHIN +-30
29AA BD 27FF JSR HI-LO * SET C IF VALUE BETWEEN LIMITS
29AD 00 30 BM $2900 * < +30
29AF 00 30 BPL $2960 * > -30
29B1 24 CD BCC $2980 * IF OUT OF LIMITS ;NEW PROBLEM
29B3  DE  5D  LDX  NUM2  * CHECK IF NUM2=+-30
29B5  DF  75  STX  VALUE  * SET C IF VALUE BETWEEN LIMITS
29B7  BD  27FF  JSR  HI-LO  * < +30
29B8  00  30  * > -30
29B9  80  30  * CHECK IF NUM2=+-30
29BB  24  C0  BCC  $2980
29BD  39  RTS

29C1  96  55  LDA  A  LOLIM1/PROBTYP  * IF MSB=1 THEN IGNORE
29C3  84  80  AND  A  #$80  * REGROUPING
29C5  26  03  BNE  #$80  * REGROUPING
29C7  39  RTS

29C8  20  B6  BRA  $2980  * EXOR SIGN BITS WITH LSB
29CA  96  50  LDA  A  OPSIGN  * OF OPSIGN, IF MSB OF A =
29CC  46  ROR  A  * 0 THEN ADDITION REGROUP
29CD  46  ROR  A  * 1 THEN SUBTRACT REGROUP
29CE  98  5B  EOR  A  NUM1
29D0  98  5D  EOR  A  NUM2
29D2  84  80  AND  A  #$80
29D4  27  3C  BEQ  $2A12
29D6  DE  5B  LDX  NUM1  * HERE FOR SUBTRACTION
29D8  DF  77  STX  UNE
29DA  DE  5D  LDX  NUM2
29DB  DF  79  STX  DEUX
29DC  70  005B  TST  NUM1
29E1  2A  08  BPL  $29EB  * BR IF SWAP NOT NEEDED
29E3  DE  5D  LDX  NUM2
29E5  DF  77  STX  UNE  * SWAP UNE & DEUX
29E7  DE  5B  LDX  NUM1  * (DEUX - UNE)
29E9  DF  79  STX  DEUX

29EB  D6  7A  LDA  B  DEUX+1  * TEST LS NIBBLE FOR
29ED  C4  0F  AND  B  #$0F  * REGROUPING
29EF  96  79  LDA  A  UNE+1
29F1  84  0F  AND  A  #$0F
29F3  0C  CLC
29F4  10  SBA
29F5  25  D1  BCS  $29C8  * TEST TENS NIBBLE
29F7  D6  7A  LDA  B  DEUX+1
29F9  C4  0F  AND  B  #$0F
29FB  96  78  LDA  A  UNE+1
29FD  84  F0  AND  A  #$0F
29FF  0C  CLC
2A00  10  SBA
2A01  25  C5  BCS  $29C8  * TEST HUNDREDS NIBBLE
2A03  D6  79  LDA  B  DEUX
2A05  C4  0F  AND  B  #$0F
2A07  96  77  LDA  A  UNE
2A09  84  0F  AND  A  #$0F
2A0B  0C  CLC
2A0C  10  SBA
2A0D  25  B9  BCS  $29C8  * IF HERE THEN NO REGROUPING
2AOF  39  RTS
185

2A10 01  NOP
2A11 01  NOP

2A12 D6 5C  LDA B  NUM1+1  *  TEST FOR ADD'N REGROUPING
2A14 C4 0F  AND B  $$OF
2A16 96 5E  LDA A  NUM2+1  *  TEST LS NIBBLE
2A18 8A 0F  AND A  $$OF
2A1A 8A 90  ORA A  $$90  *  ?  PROPAGATES CARRY
2A1C 1B  ABA
2A1D 19  DAA
2A1E 25 A8  BCS $$29CB
2A20 D6 5C  LDA B  NUM1+1  *  TEST TENS NIBBLE
2A22 C4 FO  AND B  $$FO
2A24 96 5E  LDA A  NUM2+1
2A26 8A FO  AND A  $$FO
2A28 1B  ABA
2A29 19  DAA
2A2A 25 9C  BCS $$29CB  *  TEST HUNDREDS NIBBLE
2A2C D6 5B  LDA B  NUM1
2A2E C4 0F  AND B  $$OF
2A30 96 5D  LDA A  NUM2
2A32 8A 0F  AND A  $$OF
2A34 8A 90  ORA A  $$90
2A36 1B  ABA
2A37 19  DAA
2A39 25 8E  BCS $$29CB  *  IF HERE THEN NO REGROUPING
2A3A 39  RTS

**** START OF REPORT SUBROUTINE ****

2A43 BD 27C6  JSR BLANK  *  BLANK ENTIRE DISPLAY
2A46 DE 64  LDX ATTPROBS  JSR BLANK  *  BLANK ENTIRE DISPLAY
2A49 DF 77  STX UNE  *  TOTAL=-$$WRONG=+RIGHT
2A4C 96 66  LDA A  WRONG
2A4E 8A 80  ORA A  $$80  *  CHANGE SIGN TO MINUS
2A50 97 79  STA A  DEUX
2A53 96 67  LDA A  WRONG+1
2A55 97 7A  STA A  DEUX+1
2A57 BD 27A6  JSR ADDEM  *  SIGNED BCD ADDITION ROUTINE
2A59 86 2E  LDA A  $$2E  *  'TOTAL'
2A5B BD 27F6  JSR SPKWD  *  LOAD WORD INTO SPEECH BUF.
2A5D 96 7B  LDA A  TROIX
2A5F 96 7C  LDA B  TROIX+1  *  A1=$=RIGHT
2A60 CE 0006  LDX $$0006  *  BUFFER ADDR FOR FIRST DISPLAY
2A62 BD 27CC  JSR UNPACK  *  SEPERATE NIBBLES INTO BYTES
2A64 CE 0007B  LDX $$0007B  *  ADDR OF TROIX
2A66 BD 27CF  JSR SAYNUM  *  PRONOUNCE PACKED NUM AT (X)
LDA A $121  * 'OVER'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A ATTPROBS
JSR SPKWD

LDA B ATTPROBS+1

LDX #$00A  * BUFFER ADDR OF 2ND DISPLAY
STX $00A  * SEPERATE NIBBLES INTO BYTES

LDA A #$EE  * CODE FOR DIVISION SIGN
STA A $0012  * BUFFER FOR OPERATION SIGN

LDX #$00A  * ADDR OF NO. ATTEMPTED

LDA A #$65  * 2 SECOND PAUSE
JSR LDPAUS  * LOAD PAUSE INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $127  * 'AND'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $128  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.

LDA A $28  * 'SECONDS'
JSR SPKWD  * LOAD WORD INTO SPEECH BUF.
187

2AC3 96 44 LDA A TIMER
2AC5 D6 45 LDA B TIMER+1 * ALLOWED-TIME LEFT=

* TIME TAKEN
2AC7 CE 006D LDX #$006D * ADDR OF DISPTIM
2AC9 BD 27BD JSR BISUB * UNSIGNED BINARY SUBTRACTION
2ACD DE 6D LDX DISPTIM
2ACF 0B INX
2AD0 6D STX DISPTIM
2AD2 86 00 LDA A #$00 * SUBTRACT 50 ($32) FROM TIME
2AD4 C6 32 LDA B #$32 * TAKEN TO GET SEC (DIV. BY 50)
2AD6 CE 006D LDX #$006D
2AD8 BD 27BD JSR BISUB * UNSIGNED BINARY SUBTRACTION
2AD9 24 1F BCC $2AFD * RETURN IF RESULT NEG. 
2ADE CE 006B LDX #$006B * ADDR OF SECONDS
2AE1 BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X & X+1
2AE4 DE 6B LDX SECONDS
2AE6 BC 0060 CFX #$0060
2AE9 26 0B BNE $2AF6
2AEG CE 0000 LDX #$0000
2AEE DF 6B STX SECONDS
2AF0 CE 006A LDX #$006A * ADDR OF MINUTES
2AF3 BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X & X+1
2AF4 DE 71 LDX BINSECS
2AF5 08 INX
2AF9 DF 71 STX BINSECS
2AFB 20 05 BRA $2AD2
2AFD 39 RTS

2AFE 20 C0 BRA $2AC0 * STEPPING STONE

2B00 DE 4A LDX SEED1 * TRANSFER SEEDS TO START
2B02 DF 7D STX $C00A * PROBLEM SETS
2B04 DE 62 LDX SEED2
2B06 DF 7F STX SEEDB
2B08 CE 0000 LDX #$0000
2B0A DF 68 STX SECONDS
2B0D DF 6A STX MINUTES
2B0F DF 64 STX ATTPROBS
2B11 DF 71 STX BINSECS
2B13 DF 66 STX WRONG

2B15 86 01 LDA A #$01 * SET SAY SO PROBLEMS
2B17 97 61 STA A SAY * WITHOUT ANSWERS ARE SAID.
2B19 7F 006C CLR ERROR * GENERATE NEW PROBLEM & ANS.
2B1C BD 75 BSR $2B93 * PROMOUNCE PROBLEM
2B1E 01 NOP
2B1F BD 27C0 JSR SAYDISP
2B22 01 NOP
2B23 01 NOP
2B24 01 NOP
2B25 BD 27C9 JSR DISPLAY * LOAD PROB. INTO DISPLAY
2B28 BD 27C3 JSR BLANKIT * BLANK MISSING NO. IN DISP. 
2B2C CE FFFE LDX #$FFFF * IF OPTION NOT SPEED THEN
2B2E 96 49 LDA A OPTION * DISPTIM=MAX ELSE TIMLIN. 
2B30 81 0E CMP A #$0E * =SPEED?
$2B32 26 02 BNE $2B36
$2B34 26 4C LDX TIMLIM
$2B36 26 6D STX DISPTIM
$2B38 7F 0014 CLR $0014 * PROMPT KEYBOARD
$2B3B BD 27E7 JSR GETDIG * GO GET DIGITS FROM KEYBOARD
$2B3E CE 0006 LDX $0006F * ADDR OF ANSWER
$2B41 8D BB BSR $2AFE * UPDATE TIME INFO
$2B43 01 NOP
$2B44 CE 005B LDX $005B * PUT ADDR OF CORRECT ANS.
$2B47 86 01 LDA A $001 * IN X REG
$2B49 91 52 CMP A MISSING
$2B4A 27 06 BEQ $2B33
$2B4D 4C INC A
$2B4E 08 INX
$2B4F 08 INX
$2B50 20 F7 BRA $2B49
$2B52 01 NOP
$2B53 EE 00 LDX 0xX
$2B55 9C 6F CFX ANSWER * COMPARE STUDENTS ANSW. &
$2B57 27 41 BED $2B9A * CORRECT ANSWER
$2B59 BD 27BA JSR ERROR * APPLY NEG. FEEDBACK
$2B5C 7D 004C TST ERROR * IF ERROR = 0 THEN TRY AGAIN
$2B5F 26 07 BNE $2B68
$2B61 7C 004C INC ERROR
$2B64 20 B9 BRA $2B4F * ASK AGAIN
$2B66 20 AD BRA $2B15 * STEPPING STONE TO NEW PROBLEM
$2B68 CE 0066 LDX $0066 * ADDR OF WRONG
$2B6B BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X \& X+1
$2B6E 7C 0061 INC SAY * ENABLE SAYING OF ANSWER
$2B71 86 19 LDA A $19 * 1/2 SECOND PAUSE
$2B73 BD 27F3 JSR LDPAUS * LOAD PAUSE INTO SPEECH BUF.
$2B76 BD 27C0 JSR SAYDISP * PRONOUNCE PROBLEM
$2B79 BD 27C9 JSR DISPLAY * LOAD PROB. INTO DISPLAY
$2B7C BD 27EA JSR WRT * WAIT TILL SPEECH-BUF EMPTY
$2B7F CE 000A LDX $000A * 5 SEC PAUSE
$2B82 BD 27F9 JSR SCAN * INPUT:KEYS OUTPUT:DISP. % TALK
$2B85 96 00 LDA A KEY
$2B87 81 17 CMP A $17 * KEY = REPEAT?
$2B89 27 EB BEQ $2B76
$2B8B CE 0064 LDX $0064 * ADDR OF ATTPROB
$2B8E BD 27B7 JSR BCDINC * INCREMENT (BCD) AT X \& X+1
$2B91 20 33 BRA $2BC6 * TEST FOR END CONDITIONS
$2B93 20 0A BRA $2B9F * STEPPING STONE
$2B95 01 NOP
$2B96 01 NOP
$2B97 01 NOP
$2B98 01 NOP
$2B99 01 NOP
289A BD 2805  JSR  RIGHT  * APPLY POSITIVE FEEDBACK
289D 20 CF  BRA  $2B6E  * GIVE ANSWER TO STUDENT
289F BD 27B1  JSR  GENERATE  * GENERATE PROBLEM
2BA2 BD 27A3  JSR  GETANS,  * FIND CORRECT ANSWER

* TEST FOR: N * 0 = 0
* 0 * N = 0
* 0 / N = 0

2BA5 96 52  LDA A  MISSING*
2BA7 81 03  CMP A  $#03  * MISSING NO.=3?
2BA9 26 01  BNE  $2BAC
2BAB 39  RTS

2BAC 96 50  LDA A  OPTION
2BAE 84 02  AND A  $#02
2BB0 26 01  BNE  $2B03  * BR IF MULT OR DIV
2BB2 39  RTS

2BB3 DE 5B  LDX  NUM1
2BB5 27 E8  BEQ  $2B9F  * IF NUM1=0 GET NEW PROB.
2BB7 DE 5D  LDX  NUM2
2BB9 27 E4  BEQ  $2B9F  * IF NUM2=0 GET NEW PROB.
2BBB 39  RTS

2B9C 01  NOP
2BBD 01  NOP
2BDE 01  NOP
2BDF 01  NOP
2BCC 01  NOP
2BC1 01  NOP
2BC2 01  NOP
2BC3 01  NOP
2BC4 01  NOP
2BC5 01  NOP

2BC6 96 49  LDA A  OPTION  * TEST EXIT CONDITIONS
2BC8 81 0D  CMP A  $#0D
2B9A 26 11  BNE  $2BDD  * OPTION = POWER?

2CCC DE 71  LDX  BINSECS  * BINSECS-TIMLIM=TIMER
2CEC DF 44  STX  TIMER
2CDO 96 4C  LDA A  TIMLIM
2CD2 D6 4D  LDA B  TIMLIM+1
2CD4 CE 0044  LDX  $#0044
2CDB BD 27BD  JSR  BISUB  * UNSIGNED BINARY SUBTRACTION
2CDA 24 8A  BCC  $2B66  * BR IF TIMLIMIT NOT EXCEEDED
2CDB 39  RTS

2D2D DE 4E  LDX  MAXPROBS  * TEST IF ALL PROBS DONE
2DFC 9C 64  CPX  ATTPROBS
2DE1 26 83  BNE  $2B66
2DE3 39  RTS
**** START OF ERROR SUBROUTINE ****

2C00 BD 27EA   JSR   WAIT   * WAIT TILL SPEECH-BUF EMPTY
2C03 86 01     LDA A  #01   * # OF TONE PAIRS
2C05 97 42     STA A DIG5
2C07 86 0A     LDA A  #0A   * # OF DISPLAY BLINKS
2C09 97 43     STA A DIG6
2C0B 8D 0D     BSR  #2C1A  * SOUND TONES
2C0D 7A 0042   DEC DIG5
2C10 26 00     BNE  #2C12

2C12 9D 11     BSR  #2C25  * BLINK DISPLAY
2C14 7A 0043   DEC DIG6
2C17 2A F9     BNE  #2C12
2C19 39     RTS

2C1A 86 3E     LDA A  #3E   * 'HI TONE'
2C1C BD 27F6   JSR   SPKWD  * LOAD WORD INTO SPEECH BUF.
2C1F 86 3E     LDA A  #3E   * 'HI TONE'
2C21 BD 27F6   JSR   SPKWD  * LOAD WORD INTO SPEECH BUF.
2C24 39     RTS

2C25 BD 27C6   JSR   BLANK   * BLANK ENTIRE DISPLAY
2C28 CE 0005   LDX  #0005   * 1/10 SECOND PAUSE
2C2B BD 27F9   JSR   SCAN   * INPUT:KEYS OUTPUT:DISP. & TALK
2C2E BD 27C9   JSR   DISPLAY   * LOAD PROB. INTO DISPLAY
2C31 BD 27C3   JSR   BLANKIT  * BLANK MISSING NO. IN DISP.
2C34 CE 0005   LDX  #0005   * 1/10 SECOND PAUSE
2C37 BD 27F9   JSR   SCAN   * INPUT:KEYS OUTPUT:DISP. & TALK
2C3A 39     RTS

**** START OF ADDEM SUBROUTINE ****

2C40 DE 77     LDX A   UNE   * CHECK FOR MINUS ZERO
2C42 8C 8000   CPX  #8000  * IN UNE AND DEUX
2C45 26 05     BNE  #2C4C
2C47 CE 0000   LDX  #0000
2C4A DF 77     STX A   UNE
2C4C DE 79     LDX A   DEUX
2C4E 8C 8000   CPX  #8000
2C51 26 05     BNE  #2C5B
2C53 CE 0000   LDX  #0000
2C56 DF 79     STX A   DEUX
2C58 96 77     LDA A  UNE
2C5A 9B 79     EOR A  DEUX
2C5C 2A 22  BPL  $2C80  * BR IF SIGNS SAME
2C5E CE 0077  LDX  $0077  * ADDR OF UNE
2C61 BD 48  BSR  $2CAE  * NEGATE (10'S COMP) IF NEG.
2C63 DE 9779  LDX  $0079  * ADDR OF DEUX
2C66 BD 46  BSR  $2CAE  * NEGATE (10'S COMP) IF NEG.
2C68 96 77  LDA A UNE
2C6A D6 7B  LDA B UNE+1  * SET UP FOR ADD
2C6C DE 79  LDX DEUX
2C6E DF 7B  STX TROIX
2C70 CE 007B  LDX $007B  * ADDR OF TROIX
2C73 BD 27B4  JSR BCDADD  * ADD A:8 TO LOC @ X
2C76 24 01  BCC $2C79  * IF C=1 THEN ANS. IS POS.
2C78 39  RTS
2C79 BD 23  BSR $2C9E  * NEGATE ANS. (10'S COMP)
2C7B 8A 80  ORA A $80  * SET SIGN BIT
2C7D A7 00  STA A 0,X
2C7F 39  RTS
2C80 96 77  LDA A UNE  * HERE IF SIGNS SAME
2C82 3A  PSH A
2C83 84 7F  AND A $7F  * TAKE ABS VALUE
2C85 97 77  STA A UNE
2C87 96 79  LDA A DEUX
2C89 84 7F  AND A $7F  * TAKE ABS VALUE
2C8B D6 7A  LDA B DEUX+1
2C8F DF 7B  STX TROIX  * SET UP FOR ADD
2C91 CE 007B  LDX $007B  * ADDR OF TROIX
2C94 BD 27B4  JSR BCDADD  * ADD A:8 TO LOC @ X
2C97 33  PUL B  * RECOVER SIGN
2C99 C4 80  AND B $80  * MASK ALL BUT SIGN
2C9A 19  ABA  * TFR SIGN TO ANS.
2C9B 97 7B  STA A TROIX
2C9D 39  RTS
2C9E 86 99  LDA A $99  * 10'S COMP SUBROUTINE
2CA0 A0 01  SUB A 1,X  * FORM 9'S COMP FIRST
2CA2 A7 01  STA A 1,X
2CA4 86 99  LDA A $99
2CA6 A2 00  SBC A 0,X
2CAB A7 00  STA A 0,X  * INC TO GET 10'S COMP
2CAC BD 27B7  JSR BCDINC  * INCREMENT (BCD) AT X & X+1
2CAD 39  RTS
2CAE 6D 00  TST 0,X  * IF VALUE AT X IS NEGATIVE
2CB0 2A FB  BPL $2CAB  * THEN DO 10'S COMP
2CB2 A6 00  LDA A 0,X  * ELSE RTS
2CB4 84 0F  AND A $0F
2CB6 A7 00  STA A 0,X
2CB8 20 E4  BRA $2C9E
**** START OF DADD SUBROUTINE ****

2CDF EB 01 ADD B 1+X * ADD A:B TO VALUE AT X
2CC1 E7 01 STA B 1+X
2CC3 A9 00 ADC A 0+X
2CC5 A7 00 STA A 0+X
2CC7 39 RTS

**** START OF TBLOOK SUBROUTINE ****

2CCF 96 48 LDA A MODE
2CD1 81 0C CMP A $+0C * MODE=LEVEL OF DIFF.?
2CD3 27 01 BEQ $2CD6
2CD5 39 RTS

2CD6 96 50 LDA A OPANT
2CD8 CE 280B LDX $+280B * STARTING ADDR OF ADDTBL IN LINK
2CDB 81 10 CMP A $+10 * OPERATION=ADD?
2CDD 27 08 BEQ $2CE7
2CDF 08 INX
2CE0 08 INX * X NOW POINTS TO ADDR OF SUBTBL IN LINK

2CE1 81 11 CMP A $+11 * OPERATION=SUB?
2CE3 27 02 BEQ $2CE7
2CE5 08 INX
2CE6 08 INX * POINT X AT M-DTBL ADDR
2CE7 EF 00 LDX 0+X * RETRIEVE STARTING ADDR
2CE9 DF 04 STX XSAV * OF LOOK UP TABLE
2CEB CE 0001 LDX $+0001 * INITIALIZE BCD COUNTER
2CEE DF 55 STX LOLIM1/PROBTYP

2CF0 DE 04 LDX XSAV
2CF2 EE 00 LDX 0+X * IF END OF TABLE FOUND (FFFF)
2CF4 8C FFFF CPX $+FFFF * THEN RESTART MACHINE!
2CF7 26 03 BNE $2CFD
2CF9 7E 2000 JMP $2000

2CFD DE 55 LDX LOLIM1/PROBTYP
2CFE 9C 53 CPX Level/HILIM1 * BR IF COUNTER=LEVEL
2D00 27 CE BEQ $2D10
2D02 DE 04 LDX XSAV * INC FOR NEXT PASS
2D04 08 INX
2D05 08 INX
2D06 DF 04 STX XSAV
2D08 CE 0055 LDX $+0055 * ADDR OF BCD COUNTER
2D0B 8D 27B7 JSR BCDINC * INCREMENT (BCD) AT X & X+1
2D0E 20 E0 BRA $2CFD

2D10 DE 04 LDX XSAV * IF LEVEL=COUNTER THEN
**** START OF ADDTBL ****

2E00 00 00
 00 02
 00 20
 00 22
 00 64
 00 66
 80 84
 80 48
 80 4B
 80 BB
 80 BB
 80 BB
 80 BB
 FF FF
 FF FF

**** START OF SUBTBL ****

2E80 40 00
 40 02
 40 20
 40 22
 40 44
 80 88

* THE DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT:
* [ RTXX XXXX ]
*** START OF M-DTBL ****

2F00 00 00
00 04
00 40
00 44
00 46
00 48
00 84
80 88
00 45
00 5A
00 55
00 AA
FF FF
FF FF

*** START OF BLANK SUBROUTINE ****

2F70 84 FF
2F72 CE 0006
2F75 BC 0016
2F78 26 01
2F79 39
2F7B A7 00
2F7D 08
2F7E 20 F5

*** START OF PACK SUBROUTINE ****

2FAB 96 3F

---

*R IS THE REGROUPING BIT.
*T IS SET IF THE SUBTRACTION PROBLEM
*IS TO BE GENERATED BY ADDITION AND THE
*NUMBERS MOVED SO THAT THE SUM IS LOCATED
*IN THE FIRST POSITION.

* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE

**DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT
* [ LXXX XXXX ]
* IF L IS SET THE OPERANDS ARE BETWEEN
* -30 AND +30
* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE EXCEPT THAT DIVISION IS
* DONE BY MULTIPLYING FIRST AND SECOND
* OPERANDS AND THEN MOVING THE NUMBERS
* TO FORM THE COMPLEMENTARY DIVISION

**THIS ROUTINE PUTS $FF IN
* DISPLAY BUFFER LOCATIONS

---

**RIS THE REGROUPING BIT.
*T IS SET IF THE SUBTRACTION PROBLEM
*IS TO BE GENERATED BY ADDITION AND THE
*NUMBERS MOVED SO THAT THE SUM IS LOCATED
*IN THE FIRST POSITION.

* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE

**DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT
* [ LXXX XXXX ]
* IF L IS SET THE OPERANDS ARE BETWEEN
* -30 AND +30
* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE EXCEPT THAT DIVISION IS
* DONE BY MULTIPLYING FIRST AND SECOND
* OPERANDS AND THEN MOVING THE NUMBERS
* TO FORM THE COMPLEMENTARY DIVISION

**THIS ROUTINE PUTS $FF IN
* DISPLAY BUFFER LOCATIONS

---

**RIS THE REGROUPING BIT.
*T IS SET IF THE SUBTRACTION PROBLEM
*IS TO BE GENERATED BY ADDITION AND THE
*NUMBERS MOVED SO THAT THE SUM IS LOCATED
*IN THE FIRST POSITION.

* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE

**DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT
* [ LXXX XXXX ]
* IF L IS SET THE OPERANDS ARE BETWEEN
* -30 AND +30
* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE EXCEPT THAT DIVISION IS
* DONE BY MULTIPLYING FIRST AND SECOND
* OPERANDS AND THEN MOVING THE NUMBERS
* TO FORM THE COMPLEMENTARY DIVISION

**THIS ROUTINE PUTS $FF IN
* DISPLAY BUFFER LOCATIONS

---

**RIS THE REGROUPING BIT.
*T IS SET IF THE SUBTRACTION PROBLEM
*IS TO BE GENERATED BY ADDITION AND THE
*NUMBERS MOVED SO THAT THE SUM IS LOCATED
*IN THE FIRST POSITION.

* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE

**DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT
* [ LXXX XXXX ]
* IF L IS SET THE OPERANDS ARE BETWEEN
* -30 AND +30
* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE EXCEPT THAT DIVISION IS
* DONE BY MULTIPLYING FIRST AND SECOND
* OPERANDS AND THEN MOVING THE NUMBERS
* TO FORM THE COMPLEMENTARY DIVISION

**THIS ROUTINE PUTS $FF IN
* DISPLAY BUFFER LOCATIONS

---

**RIS THE REGROUPING BIT.
*T IS SET IF THE SUBTRACTION PROBLEM
*IS TO BE GENERATED BY ADDITION AND THE
*NUMBERS MOVED SO THAT THE SUM IS LOCATED
*IN THE FIRST POSITION.

* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE

**DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT
* [ LXXX XXXX ]
* IF L IS SET THE OPERANDS ARE BETWEEN
* -30 AND +30
* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE EXCEPT THAT DIVISION IS
* DONE BY MULTIPLYING FIRST AND SECOND
* OPERANDS AND THEN MOVING THE NUMBERS
* TO FORM THE COMPLEMENTARY DIVISION

**THIS ROUTINE PUTS $FF IN
* DISPLAY BUFFER LOCATIONS

---

**RIS THE REGROUPING BIT.
*T IS SET IF THE SUBTRACTION PROBLEM
*IS TO BE GENERATED BY ADDITION AND THE
*NUMBERS MOVED SO THAT THE SUM IS LOCATED
*IN THE FIRST POSITION.

* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE

**DATA IS OF THE FORMAT:
* [ OPTION BYTE ] [ OPERAND BYTE ]
* WHERE [ OPTION BYTE ] IS OF THE FORMAT
* [ LXXX XXXX ]
* IF L IS SET THE OPERANDS ARE BETWEEN
* -30 AND +30
* THE [ OPERAND BYTE ] IS DEFINED AS IN
* ADDITION TABLE EXCEPT THAT DIVISION IS
* DONE BY MULTIPLYING FIRST AND SECOND
* OPERANDS AND THEN MOVING THE NUMBERS
* TO FORM THE COMPLEMENTARY DIVISION

**THIS ROUTINE PUTS $FF IN
* DISPLAY BUFFER LOCATIONS
2FAD 48    ASL A
2FAE 48    ASL A    * FINAL FORMAT OF PACKED DATA
2FF 48     ASL A    * /SXXX/HUND'S/ /TENS/ONES/
2FBO 48    ASL A    * WHERE S=SIGN BIT (1=MINUS)
2FB1 9A 3E  ORA A DIG1
2FB3 A7 01  STA A 1,X .
2FB5 96 3D  LDA A SGNDIG
2FB7 84 80  AND A #$80    * MASK OFF ALL BUT SIGN BIT
2FB9 9A 40  ORA A DIG3
2FBB A7 00  STA A 0,X
2FBD 7D 003D TST SGNDIG
2FC0 2A 06  BNE #$2FCB
2FC2 A6 00  LDA A 0,X    * SET SIGN BIT IF GETDIG
2FC4 8A 80  ORA A #$80    * TIMES OUT WITH NO ENTRIES
2FC6 A7 00  STA A 0,X
2FC8 39    RTS

**** START OF RIGHT SUBROUTINE ****
2FB0 86 FF  LDA A #$FF    * TURN OFF PROMPT
2FB2 97 14  STA A #$014
2FB4 86 3D  LDA A #$3D    * "LO TONE"
2FB6 BD 2786 JSR SPKWD    * LOAD WORD INTO SPKBUF
2FB8 BD 2786 JSR SPKWD    * "LO TONE"
2FBA BD 2786 JSR SPKWD    * LOAD WORD INTO SPKBUF
2FB6 BD 2786 JSR SPKWD    * "LO TONE"
2FBC BD 2786 JSR WAIT    * WAIT UNTIL SPKBUF EMPTY
2FB6 39    RTS

**** RESTART AND INTERRUPT VECTORS ****
2FF8 20 27    * INTERRUPT VECTOR
2FFA 20 00    * SWI VECTOR
2FFC 20 00    * NON-MASKABLE INTERRUPT VECTOR
2FFE 20 00    * RESTART VECTOR

**********************************************************************************************************
**** END OF CONTROL PROGRAMS ****
**********************************************************************************************************
The 2708 EPROMs were programmed using a PRO-LOG M900 Universal PROM Programmer, equipped with a model 9115 RS232 interface. The programmer was controlled by a D2 microcomputer having a 4K RAM board. The D2 kit was interfaced to the programmer via a standard 25 pin communications connector and the circuit shown in Fig. 1. After loading the 4K RAM with the data to be programmed, the EPROM burning program, shown herein, was loaded. The addresses of the beginning and end of the data block were entered into memory at locations $0000$-$0003$. The PROLOG was reset and the program was executed starting at $0004$. 
FIGURE 1 - D2 to Prolog Interface
NOTE: St. Addr. of Block $0000-01, End Addr. of Block - $0002-03

MAIN

$0004 JSR INITIALIZE BD,00,92
       LDAA,$30 86,30 "0"
       JSR OUTCR BD,00,7F "0"
       LDA $30 86,33 "F" Supply EPROM Address Range
       JSR OUTCR BD,00,7F "F"
       LDA $46 86,46 "F"
       JSR OUTCR BD,00,7F "P" (Start Program Mode)
       LDA $50 86,50
       JSR OUTCR BD,00,7F

LOOP

LDX START DE,00 Output Data
LDAAX,0 A6,00
JSR OUTBYTE BE,00,52

LDX START DE,00 Move Pointer
CPX END 9C,02
BEQ EXIT 27,05
INX 08
STX START DF,00
BRA LOOP 20,EE

EXIT

LDA $04 86,04 "EOT"
JSR OUTCR BD,00,7F
SWI 3F(at $0041)

($0042)

CMPA $09 81,09
BGT(AF) 2E,06
ADD A $30 8B,30 $0+$30=$30="0"
JSR OUTCR BD,00,7F
RTS 39

(4C)AF

ADD A $47 8B,37 $37+$0A=$41="A"
JSR OUTCR BD,007F
RTS 39 AT $0051

($0052)

OUTBYTE

PUSH A 36
LSR A 44
LSR A 44
LSR A 44
LSR A 44
JSR HEXOUT BD,00,42
PULL A 32
AND A, $0F 84,0F
JSR HEXOUT BD,00,42
RTS 39 AT $0060

($0070)

OUTCH

PUSH B 37 SEND
JSR PATCH BD,00,A4 CHARACTER WHEN
ASR B 57 PROLOG IS
ASR B 57 READY
BCC 24,F9 (i.e., CTS=+12V)
LDA B $8004 F6,80,04
ASR B 57
BCS 25,F3
STA A,$8009 B6,80,09 SEND!
PULL B 33

($0091) RTS 39

($0092) INITIALIZE LD A A,#881 86,81
ST A A,$8008 B7,80,08
LD A A#$00 86,00
ST A A,$8004 B6,80,08
LDA A,A#$04 86,04
STA A,$8005 B6,80,05

($00A1) RTS 39

($00A4) PATCH LDX $0FF CE,OF,FF
DEX 09 DELAY
BNE {PATCH} 26,FD
LDA B,$8008 F6,80,08

($00AD) RTS 39
APPENDIX V

USERS MANUAL

A. Device Overview

The Mathematics Trainer has been designed to tutor a visually handicapped student in the four basic arithmetic operations: addition, subtraction, multiplications, and division. The device uses speech in order to present the student with an incomplete mathematical sentence such as "three plus two equals blank". The student will use a numeric keypad to enter the number that should be in the blank. A sketch of the Trainer is shown in Figs. 1 and 2. The device will inform the student of the responses's correctness. If the answer is incorrect, the device will repeat the problem. If the student fails to enter the correct response the second time, the device will inform the student of the response's incorrectness and will speak the entire problem with the correct answer in its proper place. If the student's answer is correct, the device will indicate that the response is correct and speak the complete mathematical sentence with the answer in place. After each problem, the device records the correctness of the student's answer and the amount of time that it took the student to respond. After the student has completed all of the problems that the teacher has assigned, the device reports the number of correct answers, the total number of problems attempted, and the total response time saying, "TOTAL[number right] OVER[number attempted]" pausing, and then saying "[number of minutes]
FIGURE 1 - Math Trainer Layout
FIGURE 2 - Keyboard Detail. Prompt labels are in both print and braille.
AND[number of seconds] SECONDS and STOP".

The teacher is able to select how the problems will be presented to the student. Problem sets may be setup on the basis of 1) a Timed mode, with a maximum total response time, 2) a Power mode, with a fixed number of problems and unlimited time, or 3) a Speed mode where the student has a fixed amount of time per problem and a fixed number of problems in the problem set.

The teacher is also able to define the contents of the problem sets. In addition to specifying the operation involved (addition, subtraction, multiplication, or division), the teacher can delimit the problem sets using two different modes. In the Level mode, (L key), the teacher supplies the device with the level of difficulty desired. In the Range mode, (R key), the teacher supplies the upper and lower bounds on the two operands needed to define the problem. The problems are generated randomly within a given problem set. In addition, each problem set is repeatable, by pushing REPEAT after the machine says "STOP". A discussion of the choices made with regard to problem set generation, as well as Learning modes, is presented in the following sections.

B. Problem Generation

1. Range Mode

In the Range mode, the teacher defines the problem set by keying in the upper and lower limits for each of the first two numbers (operands) in the problem. The device randomly generates a number between or equal to these higher and lower limits. These random numbers become the actual values used in the problem. However,
in division, the limits refer to the second and third numbers (i.e., the divisor and the quotient).

This mode allows the teacher to tailor problem sets to a student's specific needs. If, for example, a student is having trouble solving problems involving multiplications by three, the teacher could have the device present only problems that have three as the second number. This could be accomplished by setting both of the limits of the second number equal to three.

2. Level of Difficulty Mode

In the level of difficulty mode, the teacher supplies a level number to the device; that number corresponds to a specific predefined problem set. Those problem sets are arranged so that, for a given operation, they are progressively more difficult as the level number increases.

The teacher needs only to increment the level number, after the student becomes proficient in the present level, in order to proceed sequentially to the next more difficult problem set. Of course, the teacher must train the student in the mathematical mechanics necessary for each problem set.

In selecting the problem set for each level, every effort was made to present the computational concepts in the same order as is currently employed in the elementary schools. One of the major determinants of problem set difficulty is the presence of problems that involve regrouping. Regrouping is a term to denote the occurrence of a carry in addition or a borrow in subtraction. The device will discriminate between problems that have regrouping for addition and subtraction. (The problem set definitions for addition and subtraction
are shown in TABLES 1 and 2). Because division is normally taught as the conceptual inverse of multiplication, it was decided that the division problem sets would be the inverses of the multiplication problem sets (see TABLE 3).

C. Problem and Answer Presentation

This device is designed to be used by students in elementary school; therefore, the problem sets are limited to integer arithmetic. Since integers are not closed under division, only problems that have integer quotients are generated.

Often, problems that are presented in the students' text books, have the missing number in positions other than to the right of the equals sign. Because of this, the device will allow the teacher to change the position of the missing number to correspond to any of the three positions in the problem sentence. The three positions are numbered 1, 2, 3 for left, center, right, respectively.

1. Pacing Options

The teacher may choose three options that govern how the device limits problem sessions. In the first option (the Timed option), the teacher limits the time the student works on a problem set. When the time runs out, the problem session ends. In the second option (Speed), the teacher is able to limit the amount of time per problem attempt, as well as the total number of problems. If the time per attempt runs out, the device acts as if the student had entered an incorrect answer; if the maximum number of problems have been reached the problem session ends. In the third option (Power) the limiting factor is the number of problems. The teacher limits the
TABLE I

ADDITION PROBLEM SETS

<table>
<thead>
<tr>
<th>Problem Set Level</th>
<th>Range of Operands</th>
<th>Augend</th>
<th>Addend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 to 4</td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td>2</td>
<td>0 to 4</td>
<td>5 to 9</td>
<td>5 to 9</td>
</tr>
<tr>
<td>3</td>
<td>5 to 9</td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td>4</td>
<td>5 to 9</td>
<td>5 to 9</td>
<td>5 to 9</td>
</tr>
<tr>
<td>5</td>
<td>0 to 9</td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td>6</td>
<td>0,10,20, ..., 90</td>
<td>0 to 9</td>
<td>0,10,20, ..., 90</td>
</tr>
<tr>
<td>7</td>
<td>0 to 9</td>
<td>0,10,20, ..., 90</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0,10,20, ..., 90</td>
<td>0,10,20, ..., 90</td>
<td></td>
</tr>
<tr>
<td>9*</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>10*</td>
<td>0 to 9</td>
<td>10 to 9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0 to 99</td>
<td>10 to 99</td>
<td></td>
</tr>
<tr>
<td>13*</td>
<td>19 to 99</td>
<td>10 to 99</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>10 to 99</td>
<td>10 to 99</td>
<td></td>
</tr>
<tr>
<td>15*</td>
<td>100 to 499</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>16*</td>
<td>0 to 9</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>100 to 499</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0 to 9</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>19*</td>
<td>100 to 499</td>
<td>10 to 99</td>
<td></td>
</tr>
<tr>
<td>20*</td>
<td>10 to 99</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>100 to 499</td>
<td>10 to 99</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>10 to 99</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>23*</td>
<td>100 to 499</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>199 to 499</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>100 to 499</td>
<td>-499 to 499</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>-499 to 499</td>
<td>100 to 499</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>-499 to 499</td>
<td>-499 to 499</td>
<td></td>
</tr>
</tbody>
</table>

*NOTE: Indicates that problems containing regrouping are not members of the problem set.
### TABLE II

**SUBTRACTION PROBLEM SETS**

<table>
<thead>
<tr>
<th>Problem Set</th>
<th>Range of Operands</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minuend</td>
<td>Subtrahend</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>0 to 4</td>
<td>0 to 4</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>0 to 4</td>
<td>5 to 9</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>5 to 9</td>
<td>0 to 4</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>5 to 9</td>
<td>5 to 9</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>6</strong></td>
<td>10 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td><strong>7</strong></td>
<td>0,10,20,...,90</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>8</strong></td>
<td>10 to 99</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>9</strong></td>
<td>10 to 99</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>100 to 499</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>11</strong></td>
<td>100 to 499</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>12</strong></td>
<td>100 to 499</td>
<td>10 to 99</td>
</tr>
<tr>
<td><strong>13</strong></td>
<td>100 to 499</td>
<td>10 to 99</td>
</tr>
<tr>
<td><strong>14</strong></td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td><strong>15</strong></td>
<td>0 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>16</strong></td>
<td>19 to 99</td>
<td>10 to 99</td>
</tr>
<tr>
<td><strong>17</strong></td>
<td>100 to 499</td>
<td>100 to 499</td>
</tr>
<tr>
<td><strong>18</strong></td>
<td>0 to 9</td>
<td>-9 to 9</td>
</tr>
<tr>
<td><strong>19</strong></td>
<td>-9 to 9</td>
<td>0 to 9</td>
</tr>
<tr>
<td><strong>20</strong></td>
<td>-9 to 9</td>
<td>-9 to 9</td>
</tr>
<tr>
<td><strong>21</strong></td>
<td>-499 to 499</td>
<td>-499 to 499</td>
</tr>
</tbody>
</table>

*NOTE:* Indicates the given operand ranges are actually those of the augend and addend in the complementary addition problems.

**NOTE:** Indicates that problems containing regrouping are not members of the problem set.
### TABLE III
MULTIPLICATION AND DIVISION PROBLEM SETS

<table>
<thead>
<tr>
<th>Problem Sets Level</th>
<th>Range of Operands*</th>
<th>Multiplicand</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 to 4</td>
<td>0 to 4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 to 4</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 to 9</td>
<td>0 to 4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0 to 9</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0,10,20,....,90</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0 to 9</td>
<td>0,10,20,....,90</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 to 9</td>
<td>10 to 99</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10 to 99</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10 to 30</td>
<td>10 to 30</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0 to 9</td>
<td>-9 to 9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>-9 to 9</td>
<td>0 to 9</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>-30 to 30</td>
<td>-30 to 30</td>
<td></td>
</tr>
</tbody>
</table>

*NOTE: For division the problem is generated by the multiplication of numbers within the ranges in the table. The product is then placed in dividend position.*
number of problems in the session, but the student may take any amount of time in responding to each problem. When the maximum number of problems have been completed the problem sessions ends.

2. Correct/Incorrect Answers and Repeat Key

Several standard phrases were created: When a problem has been answered correctly, a string of three low tones are said by the synthesizer; An incorrect response elicits two high tones. The words "go please", serve as a general purpose audio prompt.

The device also has the capability to repeat a problem set verbatim. The major use of this feature would be if the teacher noticed that the student had difficulty with the present problem set. The teacher could then cause the problem set to be repeated, and observe the types of problems that give the student the most difficulty.

If a sighted student forgets the current problem, that student can simply look at the problem again. A blind student listening to a verbal problem should have the same opportunity. For this reason a repeat problem presentation button has been incorporated into the device. Pushing the repeat button, while the trainer is waiting for the answer to a problem, causes the problem to be re-read.

3. Keypad and Keypad Drill

Because many students are familiar with the Speech Plus calculator, the device's keypad is designed with the numeric keys in the same configuration as the calculator. The only difference being the presence of an enter key and a minus key (for negative number entry).
The device also has a keypad training drill, which begins after the teacher has completed setting up the Math Trainer. This drill will ask the student to depress the keys on the keypad in order, saying "zero please", "one please", etc. If the student hits the correct key, that key will echo its designation and sound a tone. Wrong keys will only echo their designations. Pushing the repeat button will cause the keyboard drill to end and the problem session to begin.

D. Setting Up (Programming) the Trainer for Use

Before each problem session, the teacher will have to set up (Program) the device in the desired configuration. Parameters such as teaching mode, operation to be performed, location of missing number, number of problems, level of difficulty, etc. must be supplied to the device. Verbal asking of the required questions is not possible with the limited vocabulary of the device. Therefore, a scheme using a prompt beside a written statement (in both braille and print) of the parameter to be entered is used. The prompt is an illuminated LED next to a print and braille label. The following sequence of events takes place in setting up the device for student use:

1) Teacher turns device on.
2) Teacher chooses either Range or Level mode via R or L keys in PROB column.
3) The Pacing Option (Timed, Speed, Power) is selected via the PACE column.
4) If the Time Option was selected the teacher enters the total time limit (via the keypad) in minutes and seconds (up to 99 minutes
and 99 seconds) remembering to push ENTER after the last digit.

5) If the Speed Option was selected the teacher enters in the time limit per attempt in minutes and seconds (up to 20 minutes). [Push ENTER].

6) If the Timed Option has not been selected the teacher enters the number of problems in the problem session via the keypad. [Push ENTER].

7) The teacher keys in the operation (+, -, x, ÷).

8) If the Level Mode was selected the teacher keys in the level of difficulty number. [Push ENTER].

9) If the Range Mode was chosen the teacher enters the high and low limits of the first operand followed by the high and low limits of the second operand. [Push ENTER after each high or low limit].

The sequence of events that the student sees are shown below:

1) The trainer begins the keyboard drill. The student should press the zero key when the device says "Zero Please". The remainder of the keys are treated in a similar fashion. [NOTE: to skip the keyboard drill, push the REPEAT key.]

2) The trainer begins the problem session. In response to a problem, the student should enter his (or her) answer using the keypad, followed by pushing the ENTER key. Correct responses are followed by three low tones, after which the problem and answer are spoken. An incorrect response to the first presentation of a problem is followed by two high tones and a second, and final, presentation of that problem. A second incorrect response is followed by two high tones, after which the problem and answer are spoken.
Pushing a programming key causes the previous entries to be erased and the words "Go Please" to be said.

3) When the problem session is over the device reports the student's score and the total student response time. After the device says "STOP" the entire problem session can be repeated by pushing the Repeat button. Pushing the Restart button, at this or at any time, causes the trainer to start over at the point where the teacher chooses the Range or Level modes. Pushing any other key will cause the device to say the score again.
VITA

The author of this thesis, Michael Joseph Linnig, was born on May 23, 1956, in Louisville, Kentucky.

His elementary education was obtained at St. Clement's School and St. Timothy's School in Louisville. His secondary education was obtained at Bishop David High School, also in Louisville. He graduate from Bishop David in May, 1974.

In August, 1974, he enrolled in the Speed Scientific School of the University of Louisville, where he majored in Electrical Engineering. He received his Bachelor of Science Degree in May, 1978, and his Master of Engineering Degree in Electrical Engineering in August, 1979.