A C.A.I. PROGRAM TO AID COMPUTER SCIENCE STUDENTS
A COMPUTER ASSISTED INSTRUCTIONAL PROGRAM
TO AID SECONDARY SCHOOL STUDENTS IN
UNDERSTANDING COMPUTERS

by

GRAHAM STEEVES, B.Sc.

A Project
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Science

McMaster University
September 1980
MASTER OF SCIENCE (1980)
(Computation)


AUTHOR: Graham Steeves, B.Sc. (McMaster University)

SUPERVISOR: Professor K.A. Redish

NUMBER OF PAGES: 57
ABSTRACT

The project describes the implementation of a program to simulate a simple computer. The program is implemented on a micro-computer for portability.

A Secondary School student, in the Ontario Educational System, at about the grade 10 or 11 level may write programs for the simulated computer. The student may also simulate, interactively, some of the processes involved in executing his program.
ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor K. A. Redish for his guidance and enthusiasm during the preparation of this work.

I would also like to thank the following people: Jeanette, Tash, Kelly and Jenny for their constant support; Kathleen, Mary Ann, Mark, John, Owen, and Miss Hodd for their friendship; Joanne for her typing skill; and Jane's Cougar.
Dedicated to all of my students who have ever asked: "How does a computer work?"
# TABLE OF CONTENTS

## CHAPTER I: INTRODUCTION
1.1 Purpose and Objectives of Project .......................... 1
1.2 Bell Laboratories "Cardiac" .................................. 4
1.3 Classroom Experience ........................................ 8

## CHAPTER II: HARDWARE-SOFTWARE SELECTION
2.1 Computer choice .............................................. 9
   2.1.1 The Challenger Series .................................. 9
   2.1.2 Data General Nova 2/10 ................................ 10
   2.1.3 Radio Shack TRS 80 ...................................... 10
   2.1.4 The Commodore Pet ..................................... 11
   2.1.5 The Apple Computer .................................... 11
   2.1.6 Hewlett-Packard 2647A Graphics Terminal ......... 11
2.2 Language Choice .............................................. 13
   2.2.1 Criteria ................................................ 13
   2.2.2 Hewlett-Packard Terminal Basic ..................... 14

## CHAPTER III: DESCRIPTION OF SIMULATOR
3.1 Instruction Format ........................................... 20
3.2 Data .................................................................... 20
3.3 Instruction Set ................................................ 20
3.4 Input/Output .................................................... 25
3.5 Memory .......................................................... 25
3.6 Instruction Register .......................................... 25
3.7 Program Counter ............................................... 26
3.8 Instruction Decoder ........................................... 26
3.9 Accumulator ..................................................... 26
3.10 Sequencing ...................................................... 27
3.11 Control .......................................................... 27

## CHAPTER IV: PROGRAM OPERATION AND CURRICULUM PLACEMENT
4.1 Overview ........................................................ 28
4.2 Menu Selection ................................................ 28
4.3 Error Diagnostics .............................................. 31
4.4 Sample Programs ............................................... 32
   4.4.1 Multiplication ........................................... 32
   4.4.2 Reversing Digits ......................................... 33
   4.4.3 Double Precision ......................................... 34
   4.4.4 Subroutines ............................................... 35
4.5 Placement in the Ontario Ministry of Education Curricula .... 37
<table>
<thead>
<tr>
<th>CHAPTER V: RESULTS AND CONCLUSIONS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Project Evaluation</td>
<td>41</td>
</tr>
<tr>
<td>5.2 Areas for further development</td>
<td>42</td>
</tr>
<tr>
<td>5.3 Conclusions</td>
<td>43</td>
</tr>
</tbody>
</table>

APPENDIX A: INSTRUCTION SET 45
APPENDIX B: PROGRAM LISTING 46
REFERENCES 56
BIBLIOGRAPHY 57
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cardiac Title Page</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Cardiac Processor</td>
<td>6a</td>
</tr>
<tr>
<td>3</td>
<td>Cardiac Memory</td>
<td>6b</td>
</tr>
<tr>
<td>4</td>
<td>Computer Display for Simulator Program</td>
<td>7</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

1.1 Purpose and Objectives of Project

As a secondary school teacher, I am often asked by my bright and eager students: "How does a computer work?" They naively assume that the reply will be a simple, one sentence, answer which will explain all. Because of this repeated question over the years, and because it is a very good question - central to the study of computer science - I have attempted, as a basic goal of this project, to answer it.

The thrust of the project, therefore, is aimed at a fourteen year old student who is just beginning to study computer science. Because of this, my objective was to keep the presentation simple, perhaps overly simplistic in several instances, yet sufficiently complex to allow the student to explore more complicated aspects of computers and computing.

It is of interest to note how the Ministry of Education for the Province of Ontario put this objective in a curriculum for computer science:

"With an awareness of how a computer is organized.... the student is ready to begin programming. It is suggested that he be introduced to both high-level and low-level
languages. He does not need to master either type of language but it is important for him to understand the basic concepts of each. Low-level or machine languages have the distinct advantage of being closely associated with the way the machine actually does computations. Hence, the mysteries that so often surround high-level languages can be dispelled by an understanding of a low-level language. The instruction set need not be extensive but should include branching and testing instructions so that the power of looping can be demonstrated." (OME 1, 1970).

Another objective which I feel is necessary for any computer assisted learning program is that it be interactive. This permits the student to become involved, especially when program segments can be corrected if errors are made.

More specifically, my objective was to illustrate the following computer concepts to the beginning student:

1) The various parts of a computer including:
   a) Input and output devices
   b) Central processing unit
   c) Memory
2) Instruction Sets
3) Instruction Formats
4) Data
5) Addressing
6) Instruction Decoding
7) Sequencing
8) Loops
9) A Loader
10) Cycling
11) Assemblers
12) Compilers
13) Absolute and relative addressing

It is also of interest to note the lack of computer-assisted learning material for the subject of computer science. Teachers in other areas of study have taken advantage of the computer and produced many good instructional programs for their disciplines. As a typical example, in the 1980 catalog of Plato Courses (a collection of user written programs available from Control Data (CON, 1980)) material can be found on topics from astronomy, to basic Chinese, to Valence Electrons.

On looking closely at the material listed under computer science, most deal with computer programming in Fortran, APL, Compass, and PL/1. Only a few deal with computer structures themselves, for example: Disk Drive Fundamentals, Input/Output Supervision, and an Introduction to Computers.

Another objective of this project was to produce a program which was portable. The assumption is that with
minimal modification to the program, it will be runnable on several other computers.

Finally, I intend to use this project write up as the basis for a teacher manual which would be useful for others developing computer science courses. Thus, much of what I have written is intended for the secondary school teacher who is just beginning to teach computer science or for the teacher who has had very little exposure to computers.

1.2 Bell Laboratories "Cardiac"

In 1968, Bell Laboratories produced a device that they called Cardiac - a cardboard illustrative aid to computation. It involved pieces of cardboard which were punched out and fitted together to form a display. The operator slid the moving parts up and down to illustrate the operations of a computer. Bell Laboratories provided these aids free of charge to schools, including instructional manuals and also a Cardiac kit made of ascetate for overhead projector use.

This project is based on the Cardiac idea and attempts to implement the concepts of Cardiac as closely as possible.
A cardboard illustrative aid to computation

Bell Canada Educational Aid
Developed by
Bell Telephone Laboratories
**OP CODE**

<table>
<thead>
<tr>
<th>Code</th>
<th>Abbr.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INP</td>
<td>Input</td>
</tr>
<tr>
<td>1</td>
<td>CLA</td>
<td>Clear and add</td>
</tr>
<tr>
<td>2</td>
<td>ADD</td>
<td>Add</td>
</tr>
<tr>
<td>3</td>
<td>TAC</td>
<td>Test Accumulator contents</td>
</tr>
<tr>
<td>4</td>
<td>SFT</td>
<td>Shift</td>
</tr>
<tr>
<td>5</td>
<td>OUT</td>
<td>Output</td>
</tr>
<tr>
<td>6</td>
<td>STO</td>
<td>Store</td>
</tr>
<tr>
<td>7</td>
<td>SUB</td>
<td>Subtract</td>
</tr>
<tr>
<td>8</td>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>9</td>
<td>HRS</td>
<td>Halt and reset</td>
</tr>
</tbody>
</table>

**INSTRUCTION DECODER**

- **INSTRUCTION REGISTER**
  - 7 4 7
- **INSTRUCTION REGISTER**
  - Move slides to agree with contents of the bug's cell.

- **ACCUMULATOR TEST**
  - Move bug ahead one cell.

- **INSTRUCTION DECODER**
  - Subtract contents of cell 4 7 from accumulator.
  - Advance card.
  - Bug to cell 00.
  - Stop.

---

"cardiac" developed by David Hagelbarger
<table>
<thead>
<tr>
<th>CELL NO</th>
<th>CONTENTS</th>
<th>CELL NO</th>
<th>CONTENTS</th>
<th>CELL NO</th>
<th>CONTENTS</th>
<th>CELL NO</th>
<th>CONTENTS</th>
<th>CELL NO</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>001</td>
<td>17</td>
<td></td>
<td>34</td>
<td></td>
<td>51</td>
<td></td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>18</td>
<td></td>
<td>35</td>
<td></td>
<td>52</td>
<td></td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td></td>
<td>19</td>
<td></td>
<td>36</td>
<td></td>
<td>53</td>
<td></td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td>20</td>
<td></td>
<td>37</td>
<td></td>
<td>54</td>
<td></td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>21</td>
<td></td>
<td>38</td>
<td></td>
<td>55</td>
<td></td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td>22</td>
<td></td>
<td>39</td>
<td></td>
<td>56</td>
<td></td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>23</td>
<td></td>
<td>40</td>
<td></td>
<td>57</td>
<td></td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td>24</td>
<td></td>
<td>41</td>
<td></td>
<td>58</td>
<td></td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>25</td>
<td></td>
<td>42</td>
<td></td>
<td>59</td>
<td></td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
<td>26</td>
<td></td>
<td>43</td>
<td></td>
<td>60</td>
<td></td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>27</td>
<td></td>
<td>44</td>
<td></td>
<td>61</td>
<td></td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>28</td>
<td></td>
<td>45</td>
<td></td>
<td>62</td>
<td></td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>29</td>
<td></td>
<td>46</td>
<td></td>
<td>63</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>30</td>
<td></td>
<td>47</td>
<td></td>
<td>64</td>
<td></td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>31</td>
<td></td>
<td>48</td>
<td></td>
<td>65</td>
<td></td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>32</td>
<td></td>
<td>49</td>
<td></td>
<td>66</td>
<td></td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>33</td>
<td></td>
<td>50</td>
<td></td>
<td>67</td>
<td></td>
<td>84</td>
<td></td>
</tr>
</tbody>
</table>
1.3 Classroom Experience with Cardiac

I have personally used Cardiac in the classroom with a reasonable degree of success. The major difficulty with Cardiac is that it becomes very tedious to execute each instruction when a loop is encountered. My implementation avoids this by allowing automatic cycling at the user's discretion.
CHAPTER II
HARDWARE, SOFTWARE SELECTION

2.1 Hardware Selection

Because the simulator program was to be used in a classroom environment, I looked at computers which were being used in the classroom. As well, I had to consider what equipment was available and to which I might have reasonable access.

Although there are many kinds of equipment on the market today I considered only the following six systems:

1) The Ohio Scientific Challenger Series
2) Data General Nova Series
3) Radio Shack TRS 80 Series
4) The Commodore Pet
5) The Apple Computer
6) Hewlett-Packard 2647A Intelligent Graphics Terminal

The following discussion by no means is a full and complete comparison of the systems mentioned. I am attempting to explain why under my circumstances I chose the machine I did.

2.1.1 The Challenger Series

A number of the Challenger Series II and III's micro-computers were available to me at McMaster University. As well, disc
storage was available. Using the 6502 8K Basic appeared to put a restriction on the use of the input command. I wanted to be able to input values as well as display them anywhere on the screen. On using an input statement the cursor and prompt symbol always moved to the bottom line of the screen. The poke command did however, allow for the opposite process of displaying data on the screen. It was because of this inability to control the positioning of the input prompt on the screen, that I did not use the Challenger.

2.1.2 Data General Nova 2/10

This system is a mini-computer and was available to me at my high school in Port Elgin. I did in fact implement part of the program on this system including the graphics and processor modules. Display speed (10 c.p.s.) and access to the machine was less than satisfactory, so that work on this system was not completed.

2.1.3 Radio Shack TRS 80

The TRS 80 with disc was another possible micro-computer that could have been used. They are quite popular and a number of
these systems are appearing in schools. This system, however, was not available to me, so that it too was not used.

2.1.4 The Commodore Pet

The Pet, as well, is another micro that could have been used. Using the 40 characters per line would have necessitated considerable change in the graphics display portion of the program. Again, very popular in the classroom but not readily available to me.

2.1.5 The Apple Computer

The Apple is another very popular micro and with Applesoft Basic and auxiliary disc storage it would have been my choice of system to use. The Applesoft Basic is very extensive and with the colour graphics capabilities would have made the simulator display very appealing. This system again was not available to me, but as an aside, the Bruce County Board of Education, for whom I work, have just bought five Apples and I intend to implement my package on the Apple in the next year.

2.1.6 Hewlett-Packard 2647A Graphics Terminal

This was the micro-computer I finally decided to implement my package on. Besides
having two such terminals readily available, it offered a great number of features for the programmer which really places it in the "Cadillac" category of micros.

The following features indicate the capabilities of this machine:

- The Basic language available on the HP Terminal is very extensive using high level graphics commands, integer, floating point and string arithmetic.

- A graphics language (AGL) is an extension to Basic which offers many easy to use graphics commands.

- Hardcopy was readily available using the modem attached and routing to the Cyber at McMaster.

- Independent graphics and alphanumeric memories offer great flexibility. The graphics memory, consisting of 32K bytes of RAM, stores a 360 by 720 dot pattern for the graphics image. The alphanumeric memory offers up to 15K bytes of RAM workspace.

- Enhancement options allow characters to be displayed in half-bright, underline,
inverse video and blinking.
- Full editing capabilities including roll up or down insert, delete, next page, previous page and user defined soft keys are great aids to programming.
- Dual tape drives using the mini cartridges each with capacity for 110,000 characters of storage provided excellent program storage facilities.

2.2 **Language Choice**

2.2.1 **Criteria**

Because one of the objectives of the project was to have a program which could be run on several other commercially available micro-computers, the Basic language seemed to be an obvious choice. Some form of Basic is available on almost all micros. On the negative side, Basic is not a structured language and therefore lends itself to unstructured computer code. Also the restriction to two character variable names leaves much to be desired in terms of writing code which is readable and readily understandable.

Perhaps a much more desirable language would have been Pascal, a structured language
which lends itself nicely to structured programming. It also allows multi-character variable names which improves program readability. Several micros have Pascal available on them, and more will soon have it in the near future. Unfortunately, no such micro was available to me, thus Basic was selected as the implementation language.

2.2.2. Hewlett-Packard Terminal Basic

The Basic interpreter can be loaded from cartridge tape, and offers a wide range of Basic commands and statements. It also contains several special terminal oriented functions. These functions allow the user to control the terminal cursor and input data from the display screen.

Also graphical operations can be controlled using a special set of graphics language statements within the basic program. This special extension to Basic is called AGL - A Graphics Language, and consists of a powerful set of graphics functions that allow the user to perform graphics operations with a minimum of programming.

At this point I would like to comment
on several of the Basic and AGL commands and statements which I found very useful in implementing my program.

**Basic Commands and Statements:**

Several statements may be assigned to a single line number by separating statements on a line with the "\" (Backslash) character.

**Commands:**

**Auto (Starting line, increment)**

This is a fairly common command which allows the interpreter to generate line numbers as the program is entered.

**Extend**

Although I did not use this particular command, it could be very useful to some programmers. The extend command allows a user to add commands to the Basic interpreter.

**Merge (Filename)**

The Merge command loads a copy of the specified file into the Basic workspace. The new program is merged with any existing program. Conflicting line numbers are replaced by the new lines. This command is very useful where part of the program is on one cassette tape and part on another.
Remove STD/STDX

This command allows the user to remove certain commands from the Basic interpreter. Remove STDX provides an additional 5K Bytes of user workspace.

Set(Condition)

Allows the user to select the default mode of operation for the interpreter. Among other things, one may "set" the default data type for numeric variables.

Statements:

Call (Sub-Program Name), (Parameters)

The Call Statement allows transfer of control to a sub-program. Parameters are matched in the call and sub-statements according to position. Corresponding variables must agree in type. This statement is part of the STDX package which I removed in favour of more workspace, thus it was not available to me, however Gosub was.

Key Code (X)

Each of the keys on the keyboard has a particular code value. The code number is returned in the variable. This is a very useful command as it allows the programmer to select certain keys for user response. This
technique was used in the cycle control portion of the program. The command is not found in most Basics.

**Restore (Line Number)**

The Restore Statement resets the data pointer to the specified data statement. This is a useful statement as it allows the programmer to control data input lists arbitrarily. The line number option is usually not found in Basic languages.

**Some Built-in Functions:**

**Movcs (R,C)**

Moves the cursor to row R, column C. R and C are screen-relative co-ordinates ranging from (1, 1) to (24, 80).

**AGL Commands:**

AGL offers several types of graphics regions:

1) The logical address space which is the range of data values which may be referenced by the terminal. It may be larger than the physical limits.

2) Physical limits - define the physical display area of the terminal.

3) Graph limits - define the desired display area. This area is within
the terminal's physical limits.

Also, AGL offers the user three unit systems:
1) User Defined Units (UDU's)
2) Graphic Display Units (GDU's)
3) Metric Units

In my application I used the Default GDU's which had a range of (0,0) to (200,100).

Locate and Frame Commands

In combination, the two commands provide an efficient way to draw a rectangle in any position on the screen. The locate command specifies the display space available for plotting data. E.G.

Limit (0,200, 0, 100)

The frame function draws a box around the current region.

Several useful labeling commands are:
Lorg (Mode): allows the user to select the label-origin position: i.e. centered or right or left justified.
LDIR (Angle): sets the letter direction of labels.
CSIZE (Height, Ratio, Slant): specifies the size and aspect ratio of characters in a label.
Move (x, y): actually moves the graphics pen to the desired co-ordinate position on the screen.
Print #0; Text: is the actual labeling command used in graphics mode.
The foregoing, indicates just a few of the graphics commands available on the HP 2647A intelligent graphics terminal.
CHAPTER III

DESCRIPTION OF SIMULATOR

The simulator program displays for the user a block diagram of a computer similar to that shown in figures (2) and (3) on pages 6a and 6b. The user may then optionally interact with the simulator program to process a program written in a pseudo-machine language. (See page 7).

The following section attempts to define the various parts of this simulated computer. The chapter following gives the user a detailed description of how the simulator operates.

3.1 Instruction Format

The instruction format is three decimal digits.

The left most digit represents the operation code.

The right two digits are interpreted as the operand.

No intervening blanks should occur between digits.

3.2 Data

Data values must be integer and in the range -999 to 999 inclusive.

3.3 Instruction Set

The instruction set consists of only 10 instructions. This is consistent with the philosophy of keeping it simple for the beginning student. The instructions are as follows:
Input (INP): OP Code 0

The operand of the input instruction designates the memory cell location to which the input value is to be assigned. E.G. the instruction

052

indicates that the input value is to be assigned to memory location 52.

Clear and Add (CLA): OP Code 1

This instruction has the effect of clearing the accumulator (set it to zero) and adding to it the contents of the memory location indicated by the operand. E.G. the instruction

152

as an instruction means the accumulator is to be set to zero and the contents of memory location 52 are to be added to it.

Add (ADD): OP Code 2

This instruction adds the contents of the memory location indicated by the operand to the accumulator. E.G. the instruction

252

Means: Add the contents of memory location 52 to the contents of the accumulator.

Test Accumulator Contents (TAG): OP Code 3

This instruction represents a conditional
transfer. If the contents of the accumulator are negative then a branch to the instruction held by the memory cell location indicated by the operand of the TAC instruction. Otherwise control passes to the instruction in the next memory location.

E.G.

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>352</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
</tr>
</tbody>
</table>

If the accumulator is negative when the instruction in location 15 is executed, control passes to memory location 52. Otherwise control passes to location 16.

**Shift (SHF): OP Code 4**

The Shift instruction is the only instruction whose operand does not refer to an address in memory. Essentially it shifts the number in the accumulator to the left "x" number of places and then to the right "y" number of places. The values of "x" and "y" are specified by the second and third digits of the Shift instruction.

The following must be kept in mind when using the Shift instruction:

From the point of view of the user, the accumulator retains only 4 digits.

Digits which overflow the accumulator are irretrievable. For example, if the accumulator
contents were 456, and the instruction register contained 433, then the contents of the accumulator would be shifted left 3 places and then right 3 places:

\[
\begin{array}{ccc}
4 & 5 & 6 \\
\text{shift 3 left} & & \\
6 & 0 & 0 & 0 \\
\text{shift 3 right} & & \\
0 & 0 & 0 & 6
\end{array}
\]

Also, when a digit is moved out it is replaced by zero. For example, a four place shift left would clear the accumulator to zero.

**Output (OUT): OP Code 5**

This instruction transfers the contents of the memory location indicated by the operand of the instruction to the Output device. E.G.

\[523\]

As an instruction transfers the contents of memory location 23 to the Output list.

**Store (STO): OP Code 6**

This instruction places the contents of the accumulator into the memory location indicated by the operand. E.G.

\[623\]

As an instruction places the contents of the
accumulator into memory location 23. In fact, if the accumulator contains a number with more than 3 digits, only the 3 least significant digits are stored.

**Subtract (SUB): OP Code 7**

This instruction subtracts the contents of the memory location indicated by the operand from the accumulator. E.G.

723

As an instruction subtracts the contents of memory location 23 from the accumulator.

**Unconditional Jump (JMP): OP Code 8**

This instruction causes control to transfer to the instruction found in the memory location indicated by the operand. E.G.

823

As an instruction causes control to pass to the instruction at memory location 23.

**Halt and Reset (HRS): OP Code 9**

This instruction terminates the program and resets the program counter to the memory location indicated by the operand. E.G.

901

As an instruction halts program execution and resets the program counter to 1.
3.4 Input

Because the traditional input device was the punched card reader, input values are typed into a rectangular card in the top left corner of the screen. Each value is then listed below on the input list. Thus a user has a complete visual record of all input.

Output

Output is handled in a similar way. All values output are listed on the output list. Again, users have a complete visual record of all output.

3.5 Memory

Memory consists of 60 locations which are displayed with their contents and corresponding addresses. When a particular memory location is being used, its contents are displayed in full bright mode. This makes the program easier to find and read, and also makes the display more attractive. Where a memory location is unused, a random collection of letters and symbols is displayed in half-bright mode. The random collection of symbols is used to convey the idea that unassigned memory locations contain essentially garbage.

3.6 Instruction Register

The Instruction Register displays the instruction to be executed. This register may be controlled automatically or manually depending on the choice of the user. If cycling is in automatic mode, the instruction register automatically
displays the correct instruction to be executed. If manual mode is used, the user must type into the instruction register rectangle the correct instruction. Retyping is allowed if an error is made. A message is displayed if an error is made.

3.7 Program Counter (PC)

The Program Counter displays the address of the next instruction to be executed. Like the instruction register, the program counter may run in automatic mode or manual mode. Also, on an error retyping is allowed - prompted by an appropriate error message.

3.8 Instruction Decoder

The Instruction Decoder displays, in inverse video, a statement in English describing the meaning of the instruction which has just been fetched to the instruction register. This is done automatically with no user input required.

3.9 Accumulator

The Accumulator represents the computer's arithmetic unit. Here, numbers are added, subtracted, shifted, and tested. All accumulator operations are automatic, requiring no user input. The accumulator variable A, is declared as long (Double Precision Real) in order that shift operations may be performed on its contents. The user is aware of only 4 digits being retained.
3.10 **Sequencing**

Sequencing begins with an instruction being fetched from memory into the instruction register. The flow chart path indicated by the arrows on the display is followed. This is the way in which sequencing is controlled and displayed to the user.

3.11 **Control Unit**

The three steps required in the fetch-execute cycle are followed in the monitoring program's execution. Those steps are:

1) The instruction indicated by the program counter is fetched from memory and placed into the instruction register.

2) The number in the program counter is incremented to give the address of the next instruction to be fetched.

3) Execution of the previously fetched instruction occurs.

Cycling control, as indicated before, has two modes of operation. The user may allow the program to cycle automatically by holding the space bar down, or permit only one fetch-execute cycle per time by hitting any other key.
CHAPTER IV
PROGRAM OPERATION

4.1 Overview

The program is designed for a single user, and attempts to illustrate the operation of a computer to a novice computer science student. The student is first shown a series of program options which are graded with respect to degree of difficulty. The user selects one of these options which runs a program. By observing the program being processed on the screen and optionally interacting with the processing, the student learns a number of things about the processing of a program by a computer.

4.2 Menu Selection

4.2.1 Option #1

This option randomly selects one of three pre-written programs, automatically places it in memory and then allows the user to process the instructions individually or several at a time.

Option #1 offers very elementary programs to the user to start with. None of the randomly selected programs are identified to the user, thus forcing him to read each
instruction to discover what the program is doing. Two of the programs in option #1 are similar in that they require user input of two numbers, after which the sum or difference of the two numbers is calculated, stored, and then output.

The third program in option #1 requires the user to input a number. By using the shift and addition instructions, the product of the input numbers and the number 11 is output. The reason for this last example is pedagogical in nature. It attempts to suggest to the student ways of implementing a multiplication instruction where one does not exist. It also introduces the student to the shift instruction which is an exceptional type of instruction in that its operand does not refer to a memory address.

4.2.2 Option #2

The second choice on the menu, is also a pre-written program automatically placed into memory. The program requires no input and produces as output the numbers from 1 to 10. This program is intended to illustrate the use of the unconditional branch and the
conditional branch instructions. It should also suggest to the student other "counting programs" such as:
- counting by 2's, 3's, 5's etc.
- counting within a specific range. For example numbers between 50 and 60.
- counting a finite Fibonacci Sequence.

4.2.3 Option #3

This option is intended to be the one which most students will use once they become familiar with the instruction set. In this option a loader is displayed in the first ten memory locations and is then executed. It allows the user to place his own program into memory, starting at location 11 and then have it executed.

End of program load is signalled by entering any negative number after which execution of the loaded program begins. This end of program condition requires that any loaded program must not contain negative data. I consider this a minor restriction and in fact should offer a good challenge to the brighter student to program around.

After a program has been processed the
user is returned to the same menu selection which also includes a stop option.

4.3 Error Diagnostics

A number of diagnostic messages are provided to assist the user in de-bugging his program and in understanding program execution. These messages appear immediately below the memory window.

A number of messages occur when incorrect or invalid input is entered. These messages are just warnings and do not cause program termination. The user is asked to re-enter the data. Data input is required for:

1) Program choice in menu
2) Input values
3) PC Incrementing Update
4) Instruction Register Update

Other messages which are given are error messages which result in program termination. Error messages will be displayed under any of the following conditions:

1) Out of memory condition i.e. assignment to a memory location beyond 60.
2) Attempt to execute an unassigned memory instruction.
3) Undefined OP-Code
4) Storing into memory locations reserved for loader.
5) Accumulator Overflow.
6) Attempt to jump into memory locations reserved for loader.
7) Instruction stored in memory location reserved for loader.

4.4 Sample Programs

The purpose of this section is to present several programs which students should reasonably be expected to write and to indicate areas which a teacher could explore using the instruction set of the simulator program.

4.4.1 Multiplication

One of the first criticisms students will make of the instruction set is that there are no multiplication or division OP-Codes. The teacher could then explore ways of programming these operations and thus introduce such concepts as subroutines and firmware to the student.

The following example is a program which requires a single digit multiplier. The multiplicand may be one or two digits.

```
BC  2 digit multiplicand
xA  1 digit multiplier
Product
```
### Program:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>045</td>
<td>Input multiplicand</td>
</tr>
<tr>
<td>12</td>
<td>404</td>
<td>Clear Acc.</td>
</tr>
<tr>
<td>13</td>
<td>646</td>
<td>Store 0 (Future Sum)</td>
</tr>
<tr>
<td>14</td>
<td>047</td>
<td>Input Multiplier</td>
</tr>
<tr>
<td>15</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>710</td>
<td>Subtract 1 (From Loader Program)</td>
</tr>
<tr>
<td>17</td>
<td>647</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>323</td>
<td>Test Acc.</td>
</tr>
<tr>
<td>19</td>
<td>146</td>
<td>Add Previous Sum</td>
</tr>
<tr>
<td>20</td>
<td>245</td>
<td>Add Multiplicand</td>
</tr>
<tr>
<td>21</td>
<td>646</td>
<td>Store Revised Sum</td>
</tr>
<tr>
<td>22</td>
<td>815</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>546</td>
<td>Output Product</td>
</tr>
<tr>
<td>24</td>
<td>900</td>
<td>Halt Reset</td>
</tr>
</tbody>
</table>

#### 4.4.2 Reversing Digits

This program takes any three digit number "XYZ", and reverses the digits to output "ZYX". This type of problem is very useful in introducing the shift operation code. Typically, students can develop a two-digit shift program first, and then easily progress to writing this program.
### Program:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>029</td>
<td>Input &quot;XYZ&quot;</td>
</tr>
<tr>
<td>12</td>
<td>129</td>
<td>Clear and Add &quot;XYZ&quot;</td>
</tr>
<tr>
<td>13</td>
<td>431</td>
<td>Shift to Produce &quot;Z00&quot;</td>
</tr>
<tr>
<td>14</td>
<td>630</td>
<td>Store &quot;Z00&quot;</td>
</tr>
<tr>
<td>15</td>
<td>129</td>
<td>Clear and Add &quot;XYZ&quot;</td>
</tr>
<tr>
<td>16</td>
<td>413</td>
<td>Shift to Produce &quot;00X&quot;</td>
</tr>
<tr>
<td>17</td>
<td>230</td>
<td>Add to Produce &quot;Z0X&quot;</td>
</tr>
<tr>
<td>18</td>
<td>630</td>
<td>Store</td>
</tr>
<tr>
<td>19</td>
<td>129</td>
<td>Clear and Add &quot;XYZ&quot;</td>
</tr>
<tr>
<td>20</td>
<td>423</td>
<td>Shift to Produce &quot;00Y&quot;</td>
</tr>
<tr>
<td>21</td>
<td>410</td>
<td>Shift to Produce &quot;0Y0&quot;</td>
</tr>
<tr>
<td>22</td>
<td>230</td>
<td>Add to Produce &quot;ZYX&quot;</td>
</tr>
<tr>
<td>23</td>
<td>630</td>
<td>Store</td>
</tr>
<tr>
<td>24</td>
<td>530</td>
<td>Output</td>
</tr>
<tr>
<td>25</td>
<td>901</td>
<td>Halt</td>
</tr>
</tbody>
</table>

### 4.4.3 Double Precision

The following program illustrates a simple method of doing Double Precision arithmetic. The program adds two six digit numbers together and outputs their sum. Because the "Hardware" in our simulator is restricted to three digit numbers, the technique used here is to store the three
least significant digits in one memory location and the three most significant digits in another.

Program:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>030</td>
<td>Input Most Significant Digits</td>
</tr>
<tr>
<td>12</td>
<td>031</td>
<td>Input Least Significant Digits</td>
</tr>
<tr>
<td>13</td>
<td>032</td>
<td>Input Most Significant Digits</td>
</tr>
<tr>
<td>14</td>
<td>033</td>
<td>Input Least Significant Digits</td>
</tr>
<tr>
<td>15</td>
<td>131</td>
<td>Add Least Significant Digits</td>
</tr>
<tr>
<td>16</td>
<td>233</td>
<td>Add Least Significant Digits</td>
</tr>
<tr>
<td>17</td>
<td>633</td>
<td>Store Least Significant Digits</td>
</tr>
<tr>
<td>18</td>
<td>403</td>
<td>Shift Overflow Right</td>
</tr>
<tr>
<td>19</td>
<td>230</td>
<td>Add Most Significant Digits</td>
</tr>
<tr>
<td>20</td>
<td>232</td>
<td>Add Most Significant Digits</td>
</tr>
<tr>
<td>21</td>
<td>632</td>
<td>Store Most Significant Digits</td>
</tr>
<tr>
<td>22</td>
<td>532</td>
<td>Output Most Significant Digits</td>
</tr>
<tr>
<td>23</td>
<td>533</td>
<td>Output Least Significant Digits</td>
</tr>
<tr>
<td>24</td>
<td>901</td>
<td>Halt</td>
</tr>
</tbody>
</table>

4.4.4 Subroutines

The concept of subroutine is a profoundly important technique in programming. The calling sequences can be illustrated in an elementary form on the simulator.

The following program example uses the previous double precision program as the
subroutine. The subroutine is called twice in the main program sequence.

Program:

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>118</td>
<td>} Prepare Return Address</td>
</tr>
<tr>
<td>12</td>
<td>633</td>
<td>} Call Subroutine</td>
</tr>
<tr>
<td>13</td>
<td>820</td>
<td>} Prepare Return Address</td>
</tr>
<tr>
<td>14</td>
<td>119</td>
<td>Call Subroutine</td>
</tr>
<tr>
<td>15</td>
<td>633</td>
<td>Halt</td>
</tr>
<tr>
<td>16</td>
<td>820</td>
<td>Return Address after First Call</td>
</tr>
<tr>
<td>17</td>
<td>901</td>
<td>Return Address after Second Call</td>
</tr>
<tr>
<td>20</td>
<td>034</td>
<td>Double Precision</td>
</tr>
<tr>
<td>21</td>
<td>035</td>
<td>Subroutine for Adding</td>
</tr>
<tr>
<td>22</td>
<td>036</td>
<td>Two 6 Digit Numbers.</td>
</tr>
<tr>
<td>23</td>
<td>037</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>637</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>403</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>636</td>
<td></td>
</tr>
</tbody>
</table>
4.5 Placement in the Ontario Ministry of Education Curricula

As indicated by the Ministry of Education Circular H.S.1., 1979 - 81, there are several guidelines covering computer science. They are:


2) Elements of Computer Technology, 1970 Senior Division.


4) Informatics, Intermediate and Senior Division 1972.

All of the above curricula refer to the need to teach students a low level language, computer organization and operation. It is best summarized in the Computer Science Curriculum, 1970.

"... The Fortran statement $X=A+B/C$ sheds no light whatsoever on precisely how the computer executes this instruction. Hence, it is both interesting and enlightening to study lower-level languages because such languages will provide a general understanding of how a computer performs basic operations. Another reason for studying lower-level languages is that the most efficient programs are generally written by those who know about the machine language counterpart. These factors contribute to the student's general
understanding of the computer and to his ability to use it effectively." (OME 2,1970).

The curriculum goes on further to comment on the placement of low-level languages in a computer course. The comment therefore is very appropriate for the placement of this program in a computer course. It is as follows:

"There are arguments in favour of studying high-level languages before low-level, and conversely there are arguments in favour of the reverse sequence. For instance, some teachers find that students prefer to start using the machine quickly by introducing very simple high-level statements. Later in the course, they use the low-level language to learn what actually happens. Other teachers find that students prefer to start solving problems by using a low-level language. Later the ease of use and the relative power of the high-level language become apparent when the student starts to use it. The important point to note is that either sequence can be effectively used." (OME 2,1970).

I personally use the former approach with the student, allowing them some immediate "hands on" experience before getting into the underlying concepts.

This C.A.I. Program could be introduced after several weeks work with a high-level language. The program itself relates to topics which cover about two weeks work.
The topics which this program illustrates are listed quite completely in the previously quoted Government Curriculum.

"A study ....... will help to illustrate the following concepts:

- The relationship between the main sections of a computer; input, output, storage or memory, arithmetic unit, and control unit.
- Various hardware features such as an accumulator and instruction counter.
- Storage, divided into a finite number of parts, each capable of containing information referenced by an address.
- Stored information as either data or an instruction.
- The idea that instructions are usually executed sequentially, unless interrupted by a branch or a halt.

"It is suggested that only very simple programs be done in the low-level language. The language used should employ decimal numbers and should contain at least the following instructions:

- Input/Output: A method of reading and printing a number.
- Arithmetic: The operators +, -, x, .
- Data Movement
- Transfer or Branch: At least an unconditional
transfer, a transfer on a zero condition and a
transfer on either a positive or a negative
condition.

- Halt" (OME 2, 1970).

Thus this C.A.I. package has a very well defined
place in the Ontario Secondary School Curriculum. The
various curricula refer to several grade levels ranging
from the intermediate grade 10 to the senior grade 11 and
12 levels. The program would be very appropriate near the
beginning of an introductory course or as review material
in a second course in computer science.
CHAPTER V
RESULTS AND CONCLUSIONS

5.1 Project Evaluation

"Testing shows the presence, not the absence, of bugs." - E. W. Dijkstra.

"A debugged program is one for which you have not yet found the conditions that make it fail." - Jerry Ogdin.

And to further quote from Yourdon on the topic of anti-bugging or defensive programming:

"Within your module, you should assume that everything that can possibly go wrong will go wrong. To paraphrase one of Murphy's Laws: Even if nothing could possibly go wrong, something inevitably will. You should take this into account when you write your program."

(YOU, 1975).

The program was written with the previous comments in mind. Considerable testing was done during the writing as well as after the program was completed.

In terms of evaluating the implementation of the "Cardiac" concept on a computer, the project was successful. In fact, the project has much enhanced the Cardiac idea because it allows for automatic cycling which removes a great deal of the tedium in processing programs on Cardiac.
The one difficulty encountered in implementing the program was with the input command. To quote the HP Basic Manual: "When data is read in, the input is taken from the line containing the cursor. All characters, including non-displaying control characters, to the right of the "?" prompt are input." (HEW, 1979). This means that any characters on the same line in the alphanumeric memory will cause an error. Characters on the same line but displayed from the graphics memory do not cause an error. This difficulty was resolved by simply selecting for input lines on the display which do not have any alphanumeric data on them.

Evaluating the project for portability makes it less successful. Most of the graphics portion of the program would have to be re-written to allow it to run on another micro.

One other area of success which I feel is important, is the production of a document which would be useful for other classroom teachers. I feel that this has been accomplished and that many new teachers will find this material very useful in planning course material for computer science programs.

5.2 Areas for Further Development

There are any number of extensions and changes which could be made to make the program simulate more closely a real computer.
Additional registers would enhance the simulator considerably. This would in turn require that the instruction set be enlarged resulting in a new instruction format. The capability of doing floating point arithmetic might also be considered. Another possible suggestion might be a memory window, which would display different areas of memory depending on the memory location of the program.

5.3 Conclusions

Logically, the next step in a computer science student's education would be to study and program in a proper assembly language. It would not surprise me, to find within ten years, in the Ontario Ministry of Education Curriculum, a course in assembly language programming. I say this in all seriousness as I recall my high school mathematics teacher saying in the late 1950's that there was no way the subject of calculus could be introduced into the secondary school curriculum. He argued that it was too complex, there was no room for it in the present math courses and that there were much more important topics such as logarithms, slide rule skills and solving triangles using logs. At that time he was head of the mathematics department, had been teaching for some time and was highly respected. It is interesting to note that an introductory course in integral and differential calculus has been offered at the grade 13 level in Ontario since 1965.
In fairness to my teacher, the pocket calculator had not yet arrived and skill in using trigonometric, logarithmic and interest tables was required to do much of the mathematics in those "good" old days.

The role of computer science education at the secondary school level is just beginning to be established and depending on the public's attitude and interest in computers, it will either grow or diminish in importance.
## APPENDIX A: Instruction Set

<table>
<thead>
<tr>
<th>Number</th>
<th>Mnemonic</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INP</td>
<td>Input to Memory Cell ....</td>
</tr>
<tr>
<td>1</td>
<td>CLA</td>
<td>Clear and Add to Accumulator contents of ...</td>
</tr>
<tr>
<td>2</td>
<td>ADD</td>
<td>Add to Accumulator contents of ....</td>
</tr>
<tr>
<td>3</td>
<td>TAC</td>
<td>Test Accumulator, if Negative jump to Instruction ....</td>
</tr>
<tr>
<td>4</td>
<td>SHF</td>
<td>Shift Accumulator Contents Left then Right</td>
</tr>
<tr>
<td>5</td>
<td>OUT</td>
<td>Output Contents of ...</td>
</tr>
<tr>
<td>6</td>
<td>STR</td>
<td>Store Contents of Accumulator into Cell ....</td>
</tr>
<tr>
<td>7</td>
<td>SUB</td>
<td>Subtract Contents of ... from Accumulator</td>
</tr>
<tr>
<td>8</td>
<td>JMP</td>
<td>Jump to Cell ...</td>
</tr>
<tr>
<td>9</td>
<td>HRS</td>
<td>Halt Computer and Reset</td>
</tr>
</tbody>
</table>
APPENDIX B: Program Listing
VARIABLE LIST

A - ACCUMULATOR
C3 - COLUMN VARIABLE FOR MEMORY DISPLAY
C4 - COLUMN PARAMETER FOR PRINT SUBROUTINE
D - ACC. PARAMETER FOR PRINT SUBROUTINE
- AND RIGHT SHIFT VARIABLE
I - MEMORY CELL COUNTER
I1 - INSTRUCTION VARIABLE, INPUT VARIABLE
I2 - INPUT VARIABLE FOR INSTR. REG. AND PC
L2 - LEFT SHIFT VARIABLE
M(60,3) MEMORY ARRAY
M(i,j)-OP-CODE
M(i,j)-OP-PAND
M(i,2)-ROW MEMORY CELL DISPLAYED ON
M(i,3)-COLUMN MEMORY CELL DISPLAYED ON
P - RANDOM VARIABLE
P1 - OUTPUT LIST ROW
P2 - INPUT LIST ROW
P3 - ROW VARIABLE FOR MEMORY DISPLAY
P4 - ROW PARAMETER FOR PRINT SUBROUTINE
X - KEYBOARD CODE VARIABLE
X2 - PROGRAM CHOICE VARIABLE
LIST1-95C

**MENU SELECTION MODULE**

**SETUP MODULE**

**MEMORY CELLS**

**MARKED UNASSIGNED**

**PROGRAM OPTION**

**ADDITION PROGRAM**

**ADDITION**

**SUBTRACTION**
660 IF R<.33 THEN GOSUB 7360:REM ** SHIFT-MULTIPLICATION
680 GOSUB 7000:REM ** MEMORY DISPLAY
690 GOTO 800:REM ** GOTO PROCESSOR
695 RESTORE 600
700 DATA 5,26,1,27,2,26,6,26,1,28,7,27,6,28,3,29,8,17,0,01,0,01,0,09,9,01 ** COUNTING PROGRAM
730 PEAD M(I,0),M(I,1)
740 NEXT I
750 GOSUB 7000:REM
760 GOTO 500:REM
770 RESTORE 680:REM
780 DATA 0,09,1,09,3,11,6,11,1,04,2,16,6,04,8,01,0,00,0,01 ** LOADER PROGRAM
790 FOR I=1 TO 10:REM
800 READ M(I,0),M(I,1)
811 NEXT I
820 GOSUB 7000:REM
830 I=0
840 PRINT * PC(*) LOCATION UPDAIE MODULE *
850 ***************
860 PRINT MOVCS(M(I+1,2),M(I+1,3)-5):CHR$(27):"DA":CHR$(27):"DS" ** TD PROCESSOR
870 GOTO 820:REM ** STARTING ADDRESS - 1
880 I=16:REM
890 PRINT MOVCS(M(17,2),M(17,3)-5):CHR$(27):"DA":CHR$(27):"DS" ** COORDS FOR I/O DISPLAY
900 PRINT MOVCS(M(I+2),M(I,3)-5):"":REM ** PUCKUP PC
910 I=I+1
920 PRINT MOVCS(M(I,2),M(I,3)-5):CHR$(27):"DA":CHR$(27):"DS" ** MEMORY LOCATION CHECK MODULE *
930 ***************
940 PRINT MOVCS(22,43):"OUT OF MEMORY" ** GTC RESTART
950 GOSUB 9200:GOTO 9100:REM ** GOTO RESTART
960 GOSUB 9200:GOTO 9100:REM
970 IF I<=60 THEN 930
980 PRINT MOVCS(22,43):"OUT OF MEMORY" ** GTC RESTART
990 GOSUB 9200:GOTO 9100:REM ** GOTO RESTART
100 GOSUB 9200:GOTO 9100:REM
**F-CYCLE CONTROL MODULE**

1. **OP-CODE PROCESSOR MODULE**

2. **E-CODE PROCESSOR MODULE**

3. **ADJ MODULE**

4. **CLA MODULE**

5. **ADD MODULE**

6. **TAC MODULE**
1302 IF M(I,1) > 10 THEN 1400 1304 PRINT MOVCS(22,43); "JUMP ATTEMPTED INTO LOADER"# 1308 GOSUB 92002 GOTO 9100 1400 IF A = 0 THEN 1420 1410 GOSUB 7401 1411 PRINT MOVCS(5,15); "MOVE PC TO CELL #1M(I,1)" 1415 GOTO 1450 1420 GOSUB 7400 1422 PRINT MOVCS(5,15); "ACC = 0, MOVE TO" 1430 PRINT MOVCS(16,15); "NEXT INSTRUCTION"# 1440 GOTO 810 1450 PRINT MOVCS(M(I,2),M(I,3) - 5);# 1470 I = M(I,1) 1475 PRINT MOVCS(M(I,2),M(I,3) - 5);CHR$(27);"DA*X:CHR$(27):ZD:" 1480 GOTO 810 1490 REM ** SHIFT MODULE = 4 1500 MOVE (35, 70) 1502 L2 = M(I,1) - 1 DIV 10?D = M(I,1) - L2*10?REM ** L2 - LEFT SHIFT 1508 GOSUB 7402?REM ** D - RIGHT SHIFT 1510 PRINT MOVCS(5,15); "SHIFT ACC CONTENTS" 1520 PRINT MOVCS(5,15); "LEFT #;L2#:;RIGHT #:D" 1521 IF L2 = 0 THEN 1527 1522 FOR J = 1 TO L2 1524 A = A*10? NEXT J 1526 A = A/10000?INT((A/10000)) = INT(A*10000)?REM ** RETAIN LAST 4 DIGITS 1527 IF D = 0 THEN 1532 1528 FOR J = 1 TO D 1530 A = A*10? NEXT J 1531 A = A/10000?INT((A/10000)) = INT(A*10000)?REM ** RETAIN LAST 3 DIGITS 1532 PRINT MOVCS(11,24);# 1534 PRINT MOVCS(11,24);# 1540 IF ABS(A) < 32000 THEN 1590 1550 PRINT MOVCS(22,43); "ACCUMULATOR HAS OVERFLOWED"# 1570 GOSUB 92002 GOTO 9100 1580 PRINT MOVCS(11,24);# 1595 GOTO 910 1600 REM ** OUTPUT MODULE 1604 GOSUB 7400 1605 PRINT MOVCS(5,15); "COPY CONTENTS OF CELL"# 1606 PRINT MOVCS(5,15); "M(I,1) INTO OUTPUT DEVICE"# 1610 R1 = R1 + 1? IF R1 = 22 THEN R1 = 4 1630 II = M(M(I,1),0) + M(M(I,1),1) 1640 PRINT MOVCS(R1,39);# 1650 GOTO 910 1660 IF I > 10 THEN X2 = 0?REM 1661 IF X2 = 3 THEN 1670 1662 IF M(I,1) > 10 THEN 1670 1664 PRINT MOVCS(22,43); "LOAD INTO MEMORY AREA RESERVED FOR LOADER"# 1668 GOSUB 92002 GOTO 9100 1670 GOSUB 7400 1671 PRINT MOVCS(5,15); "COPY CONTENTS OF ACC"# 1672 PRINT MOVCS(6,15); "INTO CELL NUMBER #1M(I,1)" 1674 IF ABS(A) > 999 THEN 1742 1676 D = A?REM ** ASSIGNMENT FOR SUBROUTINE 1678 GOSUB 76002?REM ** MEMORY CELL LOAD 1720 M(M(I,1),0) = A?DIV 100?REM ** OP CODE 1730 M(M(I,1),1) = A - (A?DIV 100)*10%REM ** OPERAND 1740 GOTO 810 1742 PRINT MOVCS(22,43); "STORING RIGHT 3 DIGITS OF ACC"# 1743 D = (A?DIV 1000)*100?REM ** A - D 1744 M(M(I,1),0) = D?DIV 100 1745 M(M(I,1),1) = D - (D?DIV 100)*100?REM ** STORING RIGHT 3 DIGITS 1746 GOSUB 92002 GOTO 810 1800 REM ** SUBTRACTION MODULE >EXIT
** SUBTRACTION MODULE **

LIST

1800 REM
1810 GOSUB 7400
1820 PRINT MOVCS(5,15);"SUBTRACT CONTENTS CF"
1830 PRINT MOVCS(6,15);"FROM ACCUMULATOR"
1840 A=A-M(M(I,1),0);100+M(M(I,1),1)
1850 IF ABS(A)<=32000 THEN 1890
1860 PRINT MOVCS(22,43);"ACCUMULATOR OVERFLOWED"
1870 GOSUB 9200; GOTO 9100
1880 PRINT MOVCS(11,23);#
1890 PRINT MOVCS(11,24);A
1900 PRINT MOVCS(11,25);#"ACCUMULATOR OVERFLOWED"
1910 GOSUB 7400
1920 GOTO 810

** UNCONDITIONAL JUMP **

1920 REM
1930 IF X2=3 THEN 1930
1940 PRINT MOVCS(22,43);"MEMORY AREA RESERVED FOR LOADER"
1950 GOSUB 9200; GOTO 9100
1960 GOSUB 7400
1970 PRINT MOVCS(5,15);"MOVE PC TO CELL #H(I,1)
1980 PRINT MOVCS(M(I,1),M(I,3)-5);#
1990 IF M(I,1)>10 THEN 2000
2000 GOSUB 7400
2010 PRINT MOVCS(M(I,1),M(I,3)-5);CHR$(27);"CA*";CHR$(27);"O"
2020 GOTO 890
2030 PRINT MOVCS(22,43);"INCORECT CONTENTS, TRY AGAIN/"
2040 GOSUB 9200
2050 PRINT MOVCS(22,43);"INCORRECT CELL NUMBER, TRY AGAIN/"
2060 GOSUB 9200
2070 GOTO 6030
2080 PRINT MOVCS(22,13);# "PC MOVE WITH USER INPUT"
2090 PRINT MOVCS(22,43);# "PC MOVE WITH USER INPUT"
2100 PRINT MOVCS(22,14);# "PC MOVE WITH USER INPUT"
2110 PRINT MOVCS(22,43);# "PC MOVE UPDATE"
2120 PRINT MOVCS(22,12);# "PC MOVE UPDATE"
2130 PRINT MOVCS(22,43);"I+1"
7000 GOSUB A9000>REM MEMORY DISPLAY MODULE
7004 C$="(1,2,9)"
7005 T=1
7010 FOR C3=0 TO 76 STEP 9
7020 FOR R3=7 TO 21
7030 M(I,2)=O3
7040 M(I,1)=C3
7050 IF M(I,1)<-9 THEN GOTO 7106
7070 R=RND
7080 IF R>.66 THEN PRINT MOVCS(R3,C3):CHR$(27);"DH";CHR$(27):"ADS"
7090 IF R<.66 AND P>+.33 THEN PRINT MOVCS(R3,C3):CHR$(27);"DH";Z$;
7095 PRINT CHR$(27);"ADS"
7100 IF R<.33 THEN PRINT MOVCS(R3,C3):CHR$(27);"DH";CHR$(27):"ADS"
7105 GOTO 7122
7110 I=100*M(I,1)+M(I,1)
7110 IF I>99 THEN 7120
7110 IF I<9 THEN 7116
7111 PRINT MOVCS(R3,C3+1):I
7112 PRINT MOVCS(R3,C3):"00";
7114 GOTO 7122
7115 PRINT MOVCS(R3,C3+1)
7118 GOTO 7122
7120 PRINT MOVCS(R3,C3-1):I
7122 T=T+1
7130 NEXT R3
7140 NEXT C3
7150 RETURN
7300 RESTORE 7310>REM
7310 DATA 0,39,1,39,4,10,2,39,6,40,5,40,9,0
7320 FOR I=17 TO 23
7330 READ M(I,0),M(I,1)> NEXT I> RETURN
7390 REM
7400 PRINT MOVCS(5,15):CHR$(27);"ADB"
7410 PRINT MOVCS(4,15):CHR$(27);"ADB"
7420 RETURN
7500 REM
7501 REM
7502 REM
7503 REM
7504 REM
7505 REM
7506 REM
7507 REM
7508 REM
7509 REM
7510 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3));CHR$(27);"ADS";CHR$(27);"ADS"
7560 IF D>99 THEN 7690
7560 IF D>9 THEN 7560
7562 IF D=0 THEN 7690
7563 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)+1):O
7564 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)):"00";
7565 RETURN
7566 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)):C
7567 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)):"0x"
7568 RETURN
7569 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)-1):D
7570 RETURN
7710 IF D>99 THEN 7790
7720 IF D>9 THEN 7790
7725 IF D=0 THEN 7790
7730 PRINT MOVCS(R4,C4+1):O
7740 PRINT MOVCS(R4,C4):"0x"
7750 RETURN
7760 PRINT MOVCS(R4,C4):D
7770 PRINT MOVCS(R4,C4):O
7780 RETURN
7790 PRINT MOVCS(R4,C4-1):D
7800 RETURN

** SHIFT-MULT. MODULE **

** INSTR. DECODER ERASE **

*** FORMATTED PRINT SUBROUTINES FOR MEMORY, INSTR. REG. PC AND I/O LISTS ***
7800  PENDUP
7820  MOVEMbral:=%OFF,OFF
7830  END

*  GRAPHICS MODULE DISPLAYING BLOCK  *
*  DIAGRAM OF COMPUTER  *
*  **********************************************************

7900  LOCAL (4,7,5,30) >FRAME >REM
7910  LOCATE (0,0,0,100) >FRAME >REM
7920  LOCATE (10,10,0,100) >FRAME >REM
7930  PRINT 500:INPUT > PRINT 00:LIST
7940  LOCATE (15,15,97,95) >FRAME >REM
7950  INPUT CARD
7960  PRINT 00:INPUT > PRINT 00:CARDS
7970  LOCATE (25,50,5,20) >FRAME >REM
7980  PC
7990  FRAME
8000  MOVE (25,17)
8010  PRINT 00:INCREMENT PC
8020  PRINT 00:TO NEXT CELL
8030  REM INSTRUCTION REGISTER
8040  LOCATE (60,90,4,25)
8050  FRAME
8060  MOVE (65,1)
8070  PRINT 00:START
8080  MCVF (60,20)
8090  CSIZE (4,3,135)
8100  PRINT 00:INSTRUCTION
8110  PRINT 00:REGISTER
8120  PRINT 00:FETCH CONTENTS
8130  PRINT 00:OF PC CELL
8140  ACCUMULATOR
8150  INSTRUCTION DECODER
8160  OUTPUT
8170  MEMORY
8180  FOR I=1 TO 15
8190  PRINT MOVCS(6I,45):I> NEXT T
8200  FOR I=16 TO 30
8210  PRINT MOVCS(6I-9,53):I> NEXT T
8220  FOR I=31 TO 45
8230  PRINT MOVCS(6I-24,62):I> NEXT T
8240  FOR I=46 TO 60
8250  PRINT MOVCS(6I-39,71):I> NEXT T
8260  MOVE (10,55)
8270  CSIZE (1,1,0)
8280  PRINT 00:INSTRUCTION
PRINT :O: 
PRINT =O:0- INP 
PRINT =O:1- CLA 
PRINT =O:2- AOO 
PRINT =O:3- TAC 
PRINT =O:4- SIF 
PRINT =O:5- OUT 
PRINT =O:6- SIT 
PRINT =O:7- SUB 
PRINT =O:8- JNP 
PRINT =O:9- INP 
MOVE (40,67) 
CSIZE (1,1,0) 
FOR I=1 TO 18 
PRINT =O: 
NEXT I 
MOVE (86,6A) 
FOR I=1 TO 16 
PRINT =C: 
NEXT I 
MOVE (50,13) 
PRINT =O: 
FOR I=1 TO 10 
PRINT =O: 
NEXT I 
FOR I=1 TO 10 
PRINT =O: 
NEXT I 
REFERENCES ARE A COMPUTER SIMULATOR 
RETURN 
REM 
REM 
REM  *************************************************  
REM  * RESTART MODULE                             
REM  *************************************************  
REM 
REM 
REM  *************************************************  
REM  * CONTINUE AND ERROR MESSAGE ERASE MODULE     
REM  *************************************************  
REM 
REM 
REM  *************************************************  
REM  * HIT RETURN TO CONTINUE                      
REM  *************************************************  
REM 
REM 
REM  *************************************************  
REM  * EXIT                                      
REM  *************************************************  
REM 
REM 
REM  *************************************************  
REM  * Goto 9140                                  
REM  *************************************************  
REM 
REM 
REM  *************************************************  
REM  * Goto 9210                                  
REM  *************************************************  
REM
REFERENCES


B I B L I O G R A P H Y

Apple Catalog, Vol. 1, No. 3, 1979

Creative Computing, April 1980, Vol. 6, No. 4.


Mano, M.M. Computer System Architecture

Struble, G.W. Assembler Language Programming:

TRS-80 Microcomputer Catalog, F-191 11/5/9