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)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE: A Computer Assisted Instructional Program to Aid Secondary School Students in Understanding Computers.

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 comupter work?"

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

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. <u>The instruction set need not be extensive but should include branching and testing instructions so that the power of looping can be demonstrated</u>." (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.

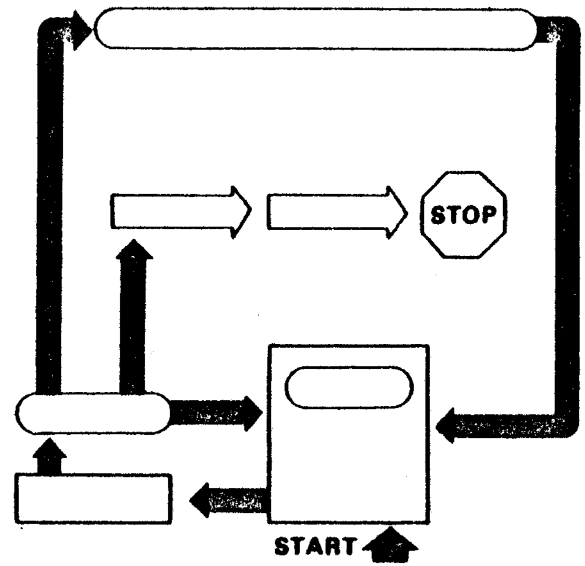
1.2 Bell Laboratories "Cardiac"

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.

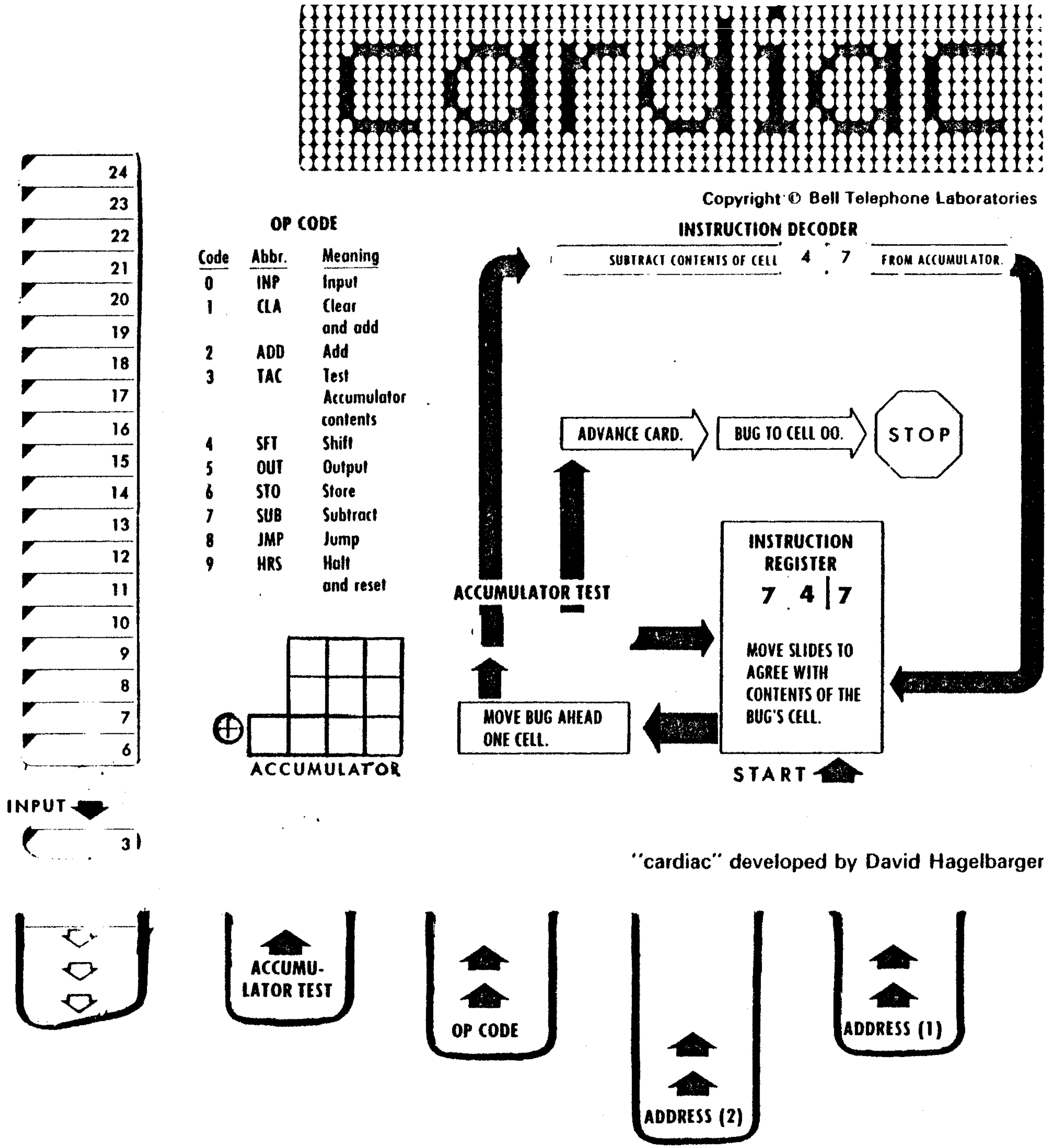
This project is based on the Cardiac idea and attempts to implement the concepts of Cardiac as closely as possible.

CARDIAC

A cardboard illustrative aid to computation



Bell Canada Educational Aid
Developed by
Bell Telephone Laboratories



"cardiac" developed by David Hagelbarger

24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8

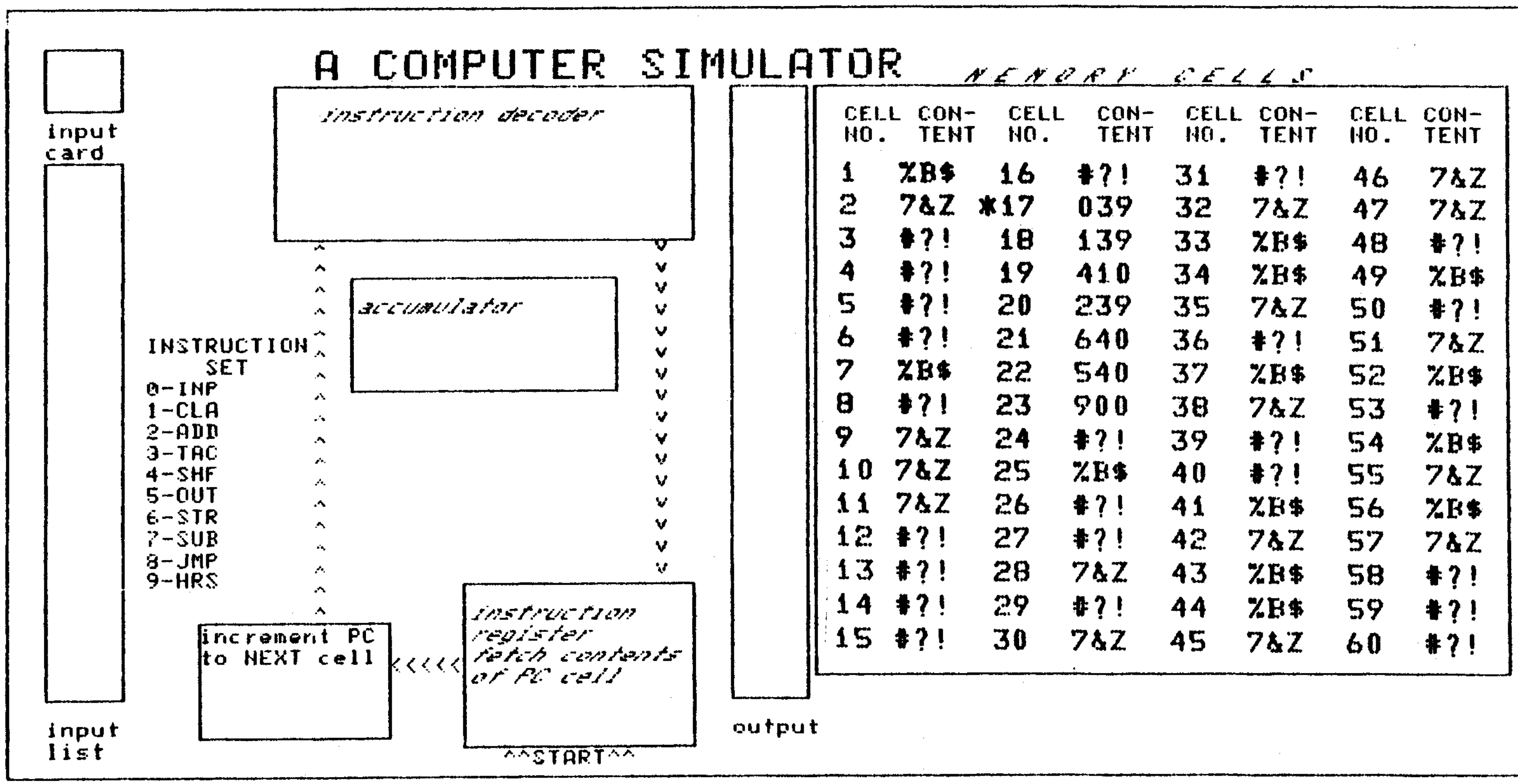
OUTPUT

5

2
1

MEMORY CELLS

CELL NO.	CONTENTS	CELL NO.	CONTENTS	CELL NO.	CONTENTS	CELL NO.	CONTENTS	CELL NO.	CONTENTS	CELL NO.	CONTENTS
00	001	17		34		51		68		85	
01		18		35		52		69		86	
02		19		36		53		70		87	
03		20		37		54		71		88	
04		21		38		55		72		89	
05				39		56		73		90	
06		23		40		57		74		91	
07		24		41		58		75		92	
08		25		42		59		76		93	
09		26		43		60		77		94	
10		27		44		61		78		95	
11		28		45		62		79		96	
12		29		46		63		80		97	
13		30		47		64		81		98	
14		31		48		65		82		99	8--
15		32		49		66		83			
16		33		50		67		84			


A COMPUTER SIMULATOR
input card
input list
instruction decoder
accumulator
INSTRUCTION SET
0-INP
1-CLA
2-ADD
3-TAC
4-SHF
5-OUT
6-STR
7-SUB
8-JMP
9-HRS
increment PC to NEXT cell
instruction register fetch contents of PC cell
^^START^^
output
MEMORY CELLS
CELL NO. CON-TENT CELL NO. CON-TENT CELL NO. CON-TENT CELL NO. CON-TENT
1 %B$ 16 #?! 31 #?! 46 7&Z
2 7&Z *17 039 32 7&Z 47 7&Z
3 #?! 18 139 33 %B$ 48 #?!
4 #?! 19 410 34 %B$ 49 %B$
5 #?! 20 239 35 7&Z 50 #?!
6 #?! 21 640 36 #?! 51 7&Z
7 %B$ 22 540 37 %B$ 52 %B$
8 #?! 23 900 38 7&Z 53 #?!
9 7&Z 24 #?! 39 #?! 54 %B$
10 7&Z 25 %B$ 40 #?! 55 7&Z
11 7&Z 26 #?! 41 %B$ 56 %B$
12 #?! 27 #?! 42 7&Z 57 7&Z
13 #?! 28 7&Z 43 %B$ 58 #?!
14 #?! 29 #?! 44 %B$ 59 #?!
15 #?! 30 7&Z 45 7&Z 60 #?!

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 <u>readily</u> 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 work-space.
- 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 (TAC): 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.

Location	Contents
15	352
16	-

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:

	4	5	6

shift 3 left

6	0	0	0

shift 3 right

0	0	0	6

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 mulitplicand may be one or two digits.

```
BC      2 digit multiplicand
xA      1 digit multiplier
        Product
```

Program:

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

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:

Address	Contents	Comments
11	029	Input "XYZ"
12	129	Clear and Add "XYZ"
13	431	Shift to Produce "Z00"
14	630	Store "Z00"
15	129	Clear and Add "XYZ"
16	413	Shift to Produce "00X"
17	230	Add to Produce "Z0X"
18	630	Store
19	129	Clear and Add "XYZ"
20	423	Shift to Produce "00Y"
21	410	Shift to Produce "0Y0"
22	230	Add to Produce "ZYX"
23	630	Store
24	530	Output
25	901	Halt

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:

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

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:

Location	Contents	Comments
11	118	Prepare Return Address
12	633	
13	820	Call Subroutine
14	119	Prepare Return Address
15	633	
16	820	Call Subroutine
17	901	Halt
18	814	Return Address after First Call
19	817	Return Address after Second Call
20	034	Double Precision Subroutine for Adding Two 6 Digit Numbers.
21	035	
22	036	
23	037	
24	135	
25	237	
26	637	
27	403	
28	234	
29	236	
30	636	

Location	Contents	Comments
31	536	
32	537	
33	860	

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:

1) Curriculum RP-33 Data Processing, 1966.

2) Elements of Computer Technology, 1970 Senior Division.

3) Computer Science, 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 is 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 comupter 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

Number	Mnemonic	Explaination
0	INP	Input to Memory Cell ....
1	CLA	Clear and Add to Accumulator contents of ...
2	ADD	Add to Accumulator contents of ....
3	TAC	Test Accumulator, if Negative jump to Instruction ....
4	SHF	Shift Accumulator Contents Left then Right
5	OUT	Output Contents of ...
6	STR	Store Contents of Accumulator into Cell ....
7	SUB	Subtract Contents of ... from Accumulator
8	JMP	Jump to Cell ...
9	HRS	Halt Computer and Reset

APPENDIX B: Program Listing

```
>LIST1-67
 1 REM
 3 REM
 5 REM ************************************************************
 7 REM *                                                          *
 9 REM *                                                          *
 11 REM *                                                         *
 13 REM *              V A R I A B L E    L I S T                 *
 15 REM *                                                         *
 17 REM *                                                         *
 19 REM *     A        - ACCUMULATOR                              *
 21 REM *     C3       - COLUMN VARIABLE FOR MEMORY DISPLAY       *
 23 REM *     C4       - COLUMN PARAMETER FOR PRINT SUBROUTINE    *
 25 REM *     D        - ACC. PARAMETER FOR PRINT SUBROUTINE      *
 26 REM *              - AND RIGHT SHIFT VARIABLE                 *
 27 REM *     I        - MEMORY CELL COUNTER                      *
 29 REM *     I1       - INSTRUCTION VARIABLE,INPUT VARIABLE      *
 31 REM *     I2       - INPUT VARIABLE FOR INSTR. REG. AND PC    *
 33 REM *     L2       - LEFT SHIFT VARIABLE                      *
 35 REM *     M(60,3)  MEMORY ARRAY                               *
 37 REM *              M(I,0)-OP-CODE                             *
 39 REM *              M(I,1)-OPERAND                             *
 41 REM *              M(I,2)-ROW MEMORY CELL DISPLAYED ON        *
 43 REM *              M(I,3)-COLUMN MEMORY CELL DISPLAYED ON     *
 45 REM *     R        - RANDOM VARIABLE                          *
 47 REM *     R1       - OUTPUT LIST ROW                          *
 49 REM *     R2       - INPUT LIST ROW                           *
 51 REM *     R3       - ROW VARIABLE FOR MEMORY DISPLAY          *
 53 REM *     R4       - ROW PARAMETER FOR PRINT SUBROUTINE       *
 55 REM *     X        - KEYBOARD CODE VARIABLE                   *
 57 REM *     X2       - PROGRAM CHOICE VARIABLE                  *
 59 REM *                                                         *
 61 REM *                                                         *
 63 REM ***********************************************************
 65 REM
 67 REM
>EXIT
```

```
>LIST1-960
 1 REM
 2 REM                    **********************************
 3 REM                    *                                *
 4 REM                    *     MENU SELECTION MODULE      *
 5 REM                    *                                *
 6 REM                    **********************************
 7 REM
 20 PLOTR ≥GPON ≥GCLR
 21 PRINT CHR$(27);≠H≠;CHR$(27);≠J≠
 22 INTEGER C3,C4,D,L2,R3,R4
 23 INTEGER M(60,3),I1,I2,R1,R2≥REM                    **SETUP MODULE
 24 CSIZE (6,1,0)≥MOVE (40,90)
 25 LONG A
 28 CSIZE (6,1,0)≥MOVE (40,90)
 30 PRINT =0;≠WELCOME TO:≠
 40 PRINT =0;≠A COMPUTER SIMULATOR∨∨≠
 50 MOVE (10,70)≥CSIZE (1,1,2)
 60 PRINT =0;≠THE FOLLOWING CHOICES OF PROGRAMS ARE AVAILABLE:≠
 70 MOVE (10,60)≥CSIZE (2,1,0)
 80 PRINT =0;≠CHOICE =1:≠
 90 MOVE (15,55)≥CSIZE (1,1,0)
 110 PRINT =0;≠PROGRAMS CONSIST OF EITHER ADDITION OR SUBTRACTION OF TWO≠
 120 PRINT =0;≠NUMBERS SUPPLIED BY THE USER OR MULTIPLICATION OF A NUMBER≠
 130 PRINT =0;≠BY 11 USING THE SHIFT AND ADD INSTRUCTIONS.≠
 140 MOVE (10,45)≥CSIZE (2,10)
 160 PRINT =0;≠CHOICE =2:≠
 170 MOVE (15,40)≥CSIZE (1,1,0)
 180 PRINT =0;≠THIS PROGRAM IS ALSO PRE-WRITTEN. IT COMPUTES AND OUTPUTS≠
 190 PRINT =0;≠THE NUMBERS FROM 1 TO 10 IN ASCENDING ORDER.≠
 200 MOVE (10,30)≥CSIZE (2,1,0)
 210 PRINT =0;≠CHOICE =3:≠
 220 MOVE (15,25)≥CSIZE (1,1,0)
 230 PRINT =0;≠THIS PROGRAM DISPLAYS A LOADER WHICH ALLOWS USERS TO LOAD≠
 240 PRINT =0;≠THEIR OWN PROGRAMS AND HAVE THEM PROCESSED.≠
 250 MOVE (10,15)≥CSIZE (6,1,0)
 260 PRINT =0;≠INPUT YOUR PROGRAM CHOICE:≠
 270 PRINT MOVC$(20,50);
 280 INPUT ≠≠,X2
 282 ON X2 GOTO 290,290,290
 284 PRINT MOVC$(23,0);≠WRONG NUMBER, TRY AGAIN∨≠
 286 GOTO 270
 290 GCLR
 400 REM
 401 REM                    ****************************************
 402 REM                    *                                      *
 403 REM                    *     PROGRAM LIBRARY SET-UP MODULE    *
 404 REM                    *                                      *
 405 REM                    ****************************************
 406 REM
 415 PRINT CHR$(27);≠H≠;CHR$(27);≠J≠
 420 REM                                                 ** MEMORY CELLS
 430 FOR I=1 TO 60>REM                                   ** MARKED UNASSIGNED
 440 M(I,0)=-9
 450 M(I,1)=-9
 460 NEXT I
 470 ON X2 GOTO 480,595,670>REM                          ** PROGRAM OPTION
 480 RESTORE 490
 490 DATA 0,36,0,37,1,36,2,37,6,38,5,38,9,01≥REM** ADDITION PROGRAM
 500 FOR I=17 TO 23
 510 READ M(I,0),M(I,1)
 520 NEXT I
 530 R=RND
 540 IF R>=.66 THEN 580≥REM                              ** ADDITION
 550 IF R<.66 AND R>=.33 THEN M(20,0)=7≥REM              ** SUBTRACTION
```

```
560 IF R<.33 THEN GOSUB 7360>REM                          ** SHIFT-MULTIPLICATION
580 GOSUB 7000>REM                                        ** MEMORY DISPLAY
590 GOTO 800>REM                                          ** GOTO PROCESSOR
595 RESTORE 600
600 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
620 FOR I=17 TO 29>REM                                    ** COUNTING PROGRAM
630 READ M(I,0),M(I,1)
640 NEXT I
660 GOSUB 7000>REM                                        ** MEMORY DISPLAY
665 GOTO 800>REM                                          ** GOTO PROCESSOR
670 RESTORE 680>REM                                       ** LOADER
680 DATA 0,09,1,09,3,11,6,11,1,04,2,10,6,04,8,01,0,00,0,01
710 FOR I=1 TO 10>REM                                     ** LOADER PROGRAM
720 READ M(I,0),M(I,1)
730 NEXT I
731 GOSUB 7000>REM                                        ** TO MEMORY DISPLAY
732 I=0
733 REM
734 REM                   ************************************
735 REM                   *  PC(*) LOCATION UPDATE MODULE  *
736 REM                   ************************************
737 REM
752 PRINT MOVCS(M(I+1,2),M(I+1,3)-5):CHR$(27);≠^DA*≠:CHR$(27);≠^D≤≠
760 GOTO 802>REM                                           ** TC PROCESSOR
800 I=16>REM                                               ** STARTING ADDRESS - 1
801 PRINT MOVCS(M(17,2),M(17,3)-5):CHR$(27);≠^DA*≠;CHR$(27);≠^D≤≠
802 R1=3
804 R2=5>REM                                               ** COORDS FOR I/O DISPLAY
808 IF I=0 THEN 812
810 PRINT MOVCS(M(I,2),M(I,3)-5);≠ ≠>REM                  ** RUBOUT PC
812 I=I+1
814 PRINT MOVCS(M(I,2),M(I,3)-5);CHR$(27);≠^DA*≠;CHR$(27);≠^D≤≠
880 REM
881 REM                    **************************************
882 REM                    *    MEMORY LOCATION CHECK MODULE    *
883 REM                    **************************************
884 REM
890 IF I<=60 THEN 930
900 PRINT MOVCS(22,43);≠OUT OF MEMORY≠
920 GOSUB 9200> GOTO 9100>REM                              ** GOTO RESTART
930 IF M(I,1)<>-9 THEN 970
940 PRINT MOVCS(22,43):≠MEMORY CELL CONTENTS UNASSIGNED≠
960 GOSUB 9200> GOTO 9100>REM                              ** GOTO RESTART
>EXIT
```

```
>LIST961-1800
 961 REM
 962 REM                          ************************************
 963 REM                          *      F-E CYCLE CONTROL MODULE     *
 964 REM                          ************************************
 965 REM
 970 GETKBD ON
 971 PRINT MOVCS(22,44);≠FOR AUTOMATIC: HIT SPACE BAR≠
 972 PRINT MOVCS(23,44);≠FOR USER INPUT: HIT ANY OTHER KEY≠
 973 IF GETKBD(X)=0 THEN 973
 975 PRINT MOVCS(22,44);≠                              ≠
 976 PRINT MOVCS(23,44);≠                                   ≠
 980 IF X=32 THEN GOSUB 6200≥REM                  ** AUTOMATIC CYCLING
 982 IF X<>32 THEN GOSUB 6000≥REM                 ** CYCLING WITH USER INPUT
 983 GETKBD OFF
 984 REM
 985 REM                         ************************************
 986 REM                         *      OP-CODE PROCESSOR MODULE     *
 987 REM                         ************************************
 988 REM
 990 ON M(I,0)+1 GOTO 1030,1130,1240,1380,1490,1600,1660,1800,1920,1990
 1000 PRINT MOVCS(22,43);≠OP-CODE NOT DEFINED≠
 1020 GOSUB 9200≥ GOTO 9100
 1030 REM                                          ** INPUT MODULE = 0
 1031 IF X2=3 THEN 1040
 1032 IF M(I,1)>10 THEN GOTO 1040
 1034 PRINT MOVCS(22,43);≠ATTEMPTED LOAD INTO AREA RESERVED FOR LOADER≠
 1038 GOSUB 9200≥ GOTO 9100
 1040 GOSUB 7400
 1042 PRINT MOVCS(5,15);≠COPY DATA FROM INPUT  ≠
 1050 PRINT MOVCS(6,15);≠DEVICE TO CELL≠;M(I,1)
 1060 R2=R2+1≥ IF R2=22 THEN R2=6
 1070 PRINT MOVCS(3,1);≠          ≠
 1072 PRINT MOVCS(3,2);
 1074 INPUT I1
 1090 IF ABS(I1)<=999 THEN GOTO 1121
 1100 PRINT MOVCS(22,43);≠INPUT VALUE OUT OF RANGE RE-ENTER≠
 1110 GOSUB 9200
 1120 GOTO 1070
 1121 M(M(I,1),0)=I1 DIV 100
 1122 M(M(I,1),1)=I1-(I1 DIV 100)*100
 1124 D=I1
 1126 GOSUB 7600
 1127 R4=R2≥C4=4≥D=I1≥ GOSUB 7710≥REM              ** PRINT FORMAT MODULE
 1128 GOTO 810
 1130 REM                                          ** CLA MODULE = 1
 1140 GOSUB 7400
 1142 PRINT MOVCS(5,15);≠COPY CONTENTS OF CELL ≠
 1150 PRINT MOVCS(6,15);M(I,1);≠ INTO ACC.     ≠
 1160 A=M(M(I,1),0)*100+M(M(I,1),1)
 1165 PRINT MOVCS(11,23);≠        ≠
 1170 PRINT MOVCS(11,24);A
 1180 GOTO 810
 1240 REM                                          ** ADD MODULE = 2
 1250 GOSUB 7400
 1252 PRINT MOVCS(5,15);≠ADD CONTENTS OF CELL  ≠
 1260 PRINT MOVCS(6,15);M(I,1);≠TO ACC. ≠
 1270 A=A+M(M(I,1),0)*100+M(M(I,1),1)
 1280 IF ABS(A)<=32000 THEN 1320
 1290 PRINT MOV(S(22,43);≠ACCUMULATOR CONTENTS OVERFLOWED≠
 1310 GOSUB 9200≥ GOTO 9100
 1320 PRINT MOVCS(11,23);≠        ≠
 1322 PRINT MOVCS(11,24);A
 1340 GOTO 810
 1380 REM                                          ** TAC MODULE = 3
```

```
1382 IF M(I,1)>10 THEN 1400
1384 PRINT MOVCS(22,43);≠JUMP ATTEMTED INTO LOADER≠
1388 GOSUB 9200≥ GOTO 9100
1400 IF A>=0 THEN 1420
1410 GOSUB 740(
1411 PRINT MOVCS(5,15);≠MOVE PC TO CELL  ≠;M(I,1)
1415 GOTO 1450
1420 GOSUB 7400
1422 PRINT MOVCS(5,15);≠ACC>=0, MOVE TO        ≠
1430 PRINT MOVCS(6,15);≠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$(27);≠^DA*≠;CHR$(27);≠^D≤≠
1480 GOTO 890
1490 REM                                          ** SHIFT MODULE = 4
1500 MOVE (35,70)
1502 L2=M(I,1) DIV 10≥D=M(I,1)-L2*10≥REM           ** L2 - LEFT SHIFT
1508 GOSUB 7400>REM                               ** D  - RIGHT SHIFT
1510 PRINT MOVCS(5,15);≠SHIFT ACC CONTENTS    ≠
1520 PRINT MOVCS(6,15);≠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)≥A=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/1000-INT(A/1000)>A=INT(A*1000)≥REM      ** RETAIN LAST 3 DIGITS
1532 PRINT MOVCS(11,23);≠        ≠
1534 PRINT MOVCS(11,24);A
1540 IF ABS(A)<=32000 THEN 1580
1550 PRINT MOVCS(22,43);≠ACCUMULATOR HAS OVERFLOWED≠
1570 GOSUB 9200≥ GOTO 9100
1580 PRINT MOVCS(11,24);A
1595 GOTO 810
1600 REM                                         ** OUTPUT MODULE
1604 GOSUB 7400
1605 PRINT MOVCS(5,15);≠COPY CONTENTS OF CELL ≠
1606 PRINT MOVCS(6,15);M(I,1);≠INTO OUTPUT DEVICE≠
1610 R1=R1+1≥ IF R1=22 THEN R1=4
1630 I1=M(M(I,1),0)*100+M(M(I,1),1)
1640 PRINT MOVCS(R1,39);I1
1650 GOTO 810
1660 IF I>10 THEN X2=0≥REM                       ** STORE MODULE
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 9200> GOTO 9100
1670 GOSUB 7400
1671 PRINT MOVCS(5,15);≠COPY CONTENTS OF ACC  ≠
1672 PRINT MOVCS(6,15);≠INTO CELL NUMBER ≠;M(I,1)
1674 IF ABS(A)>999 THEN 1742
1676 D=A≥REM                                     ** ASSIGNMENT FOR SUBROUTINE
1678 GOSUB 7600≥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)*100≥REM           ** OPERAND
1740 GOTO 810
1742 PRINT MOVCS(22,43);≠STORING RIGHT 3 DIGITS OF ACC.≠
1743 D=(A DIV 1000)*1000≥D=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 9200≥ GOTO 810
1800 REM                                         ** SUBTRACTION MODULE
>EXIT
```

```
>LIST1800-7800
 1800 REM                                                  ** SUBTRACTION MODULE
 1808 GOSUB 7400
 1810 PRINT MOVCS(5,15);≠SUBTRACT CONTENTS OF≠
 1820 PRINT MOVCS(6,15);M(I,1);≠ 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≠
 1880 GOSUB 9200≥ GOTO 9100
 1890 PRINT MOVCS(11,23);≠        ≠
 1895 PRINT MOVCS(11,24);A
 1910 GOTO 810
 1920 IF I>10 THEN X2=0≥REM                               ** UNCONDITIONAL JUMP
 1921 IF X2=3 THEN 1930
 1922 IF M(I,1)>10 THEN 1930
 1924 PRINT MOVCS(22,43);≠MEMORY AREA RESERVED FOR LOADER≠
 1928 GOSUB 9200≥ GOTO 9100
 1930 GOSUB 7400
 1932 PRINT MOVCS(5,15);≠MOVE PC TO CELL     ≠;M(I,1)
 1950 PRINT MOVCS(M(I,2),M(I,3)-5);≠ ≠
 1970 I=M(I,1)
 1975 PRINT MOVCS(M(I,2),M(I,3)-5);CHR$(27);≠^CA*≠;CHR$(27);≠^D<≠
 1980 GOTO 890
 1990 REM                                                  ** HALT AND STOP MODULE
 2000 GOSUB 7400
 2002 PRINT MOVCS(5,15);≠STOP COMPUTER≠
 2030 GOTO 9100
 5990 REM                    ****************************************
 5991 REM                    *       USER CYCLE CONTROL MODULE      *
 5992 REM                    ****************************************
 5993 REM
 6000 REM                                       INSTR. REG. UPDATE
 6010 REM                                       WITH USER INPUT
 6020 I1=M(I,0)*100+M(I,1)
 6022 PRINT MOVCS(23,26);≠        ≠
 6030 PRINT MOVCS(23,27);
 6040 INPUT I2
 6050 IF I1=I2 THEN 6090
 6060 PRINT MOVCS(22,43);≠INCORRECT CONTENTS,TRY AGAIN∨≠
 6070 GOSUB 9200
 6080 GOTO 6030
 6090 REM
 6100 REM                                       PC MOVE WITH USER INPUT
 6105 PRINT MOVCS(22,13);≠        ≠
 6110 PRINT MOVCS(22,14);
 6120 INPUT I2
 6130 IF I2=I+1 THEN 6180
 6140 PRINT MOVCS(22,43);≠INCORRECT CELL NUMBER, TRY AGAIN∨≠
 6150 GOSUB 9200
 6160 GOTO 6110
 6180 RETURN
 6200 REM                                       INSTR. REG. UPDATE
 6220 I2=M(I,0)*100+M(I,1)
 6225 PRINT MOVCS(23,25);≠      ≠
 6230 D=I2≥R4=23≥C4=29
 6232 GOSUB 7710≥REM                                  PRINT FORMAT MODULE
 6240 REM                                       PC MOVE UPDATE
 6245 PRINT MOVCS(22,12);≠      ≠
 6250 PRINT MOVCS(22,14);I+1
 6270 RETURN
 6990 REM
 6991 REM                    *******************************************
 6992 REM                    *      MEMORY CONTENTS DISPLAY MODULE     *
 6993 REM                    *******************************************
 6994 REM
```

```
7000 GOSUB 8000>REM MEMORY DISPLAY MODULE
7004 CSIZE (1,2,0)
7005 I=1
7010 FOR C3=49 TO 76 STEP 9
7020 FOR R3=7 TO 21
7030 M(I,2)=R3
7040 M(I,3)=C3
7060 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);≠^D≤≠
7090 IF R<.66 AND R>=.33 THEN PRINT MOVCS(R3,C3);CHR$(27);≠^DH7^Z≠;
7095 PRINT CHR$(27);≠^D≤≠
7100 IF R<.33 THEN PRINT MOVCS(R3,C3);CHR$(27);≠^DH%B$≠;CHR$(27);≠^D≤≠
7105 GOTO 7122
7106 I1=100*M(I,0)+M(I,1)
7108 IF I1>99 THEN 7120
7110 IF I1>9 THEN 7116
7111 PRINT MOVCS(R3,C3+1);I1
7112 PRINT MOVCS(R3,C3);≠00≠
7114 GOTO 7122
7116 PRINT MOVCS(R3,C3);I1
7117 PRINT MOVCS(R3,C3);≠0≠
7118 GOTO 7122
7120 PRINT MOVCS(R3,C3-1);I1
7122 I=I+1
7130 NEXT R3
7140 NEXT C3
7150 RETURN
7300 RESTORE 7310>REM                                  ** SHIFT-MULT. MODULE
7310 DATA 0,39,1,39,4,10,2,39,6,40,5,40,9,0
7320 FOR I=17 TO 23
7330 READ M(I,0),M(I,1)> NEXT I> RETURN
7390 REM                                               ** INSTR. DECODER ERASE
7400 PRINT MOVCS(5,15);CHR$(27);≠^DB                           ≠;CHR$(27);≠^D≤≠
7410 PRINT MOVCS(6,15);CHR$(27);≠^DB                           ≠;CHR$(27);≠^D≤≠
7420 RETURN
7590 REM
7591 REM                    *******************************************
7592 REM                    *   FORMATTED PRINT SUBROUTINES FOR MEMORY, *
7593 REM                    *  INSTR. REG. PC AND I/O LISTS             *
7594 REM                    *******************************************
7595 REM
7600 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3));CHR$(27);≠^D≤   ≠;CHR$(27);≠^D≤≠
7610 IF D>99 THEN 7690
7620 IF D>9 THEN 7660
7625 IF D<0 THEN 7690
7630 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)+1);D
7640 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3));≠00≠
7650 RETURN
7660 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3));D
7670 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3));≠0≠
7680 RETURN
7690 PRINT MOVCS(M(M(I,1),2),M(M(I,1),3)-1);D
7700 RETURN
7710 IF D>99 THEN 7790
7720 IF D>9 THEN 7760
7725 IF D<0 THEN 7790
7730 PRINT MOVCS(R4,C4+1);D
7740 PRINT MOVCS(R4,C4);≠00≠
7750 RETURN
7760 PRINT MOVCS(R4,C4);D
7770 PRINT MOVCS(R4,C4);≠0≠
7780 RETURN
7790 PRINT MOVCS(R4,C4-1);D
7800 RETURN
```

```
7800 RETURN
7990 REM
7991 REM                    ***************************************************
7992 REM                    *      GRAPHICS MODULE DISPLAYING BLOCK           *
7993 REM                    *            DIAGRAM OF COMPUTER                  *
7994 REM                    ***************************************************
7995 REM
8000 LOCATE (0,200,0,100)≥FRAME ≥REM                        ** DISPLAY
8010 LOCATE (5,15,10,80)≥FRAME ≥REM                         ** INPUT
8020 MOVE (5,5)≥CSIZE (1,2,0)≥ PRINT ≡0:≠INPUT≠≥ PRINT ≡0:≠LIST≠
8030 LOCATE (5,15,87,95)≥FRAME ≥REM                         ** INPUT CARD
8050 MOVE (5,83)≥ PRINT ≡0:≠INPUT≠≥ PRINT ≡0:≠CARD≠
8120 LOCATE (25,50,5,20)≥FRAME ≥REM                         ** PC
8130 FRAME
8140 MOVE (25,17)
8150 PRINT ≡0:≠INCREMENT PC≠
8155 PRINT ≡0:≠TO NEXT CELL≠
8160 REM INSTRUCTION REGISTER
8170 LOCATE (60,90,4,25)
8180 FRAME
8190 MOVE (65,1)
8210 PRINT ≡0:≠""START""≠
8220 MOVE (60,20)
8230 CSIZE (4,3,135)
8240 PRINT ≡0:≠INSTRUCTION≠
8245 PRINT ≡0:≠REGISTER≠≥ PRINT ≡0:≠FETCH CONTENTS≠≥ PRINT ≡0:≠OF PC CELL≠
8250 REM                                                    ** ACCUMULATOR
8260 LOCATE (45,80,50,65)
8270 FRAME
8280 MOVE (45,60)
8290 CSIZE (1,1,90)
8300 PRINT ≡0:≠ACCUMULATOR≠
8310 REM                                                    ** INSTRUCTION DECODER
8320 LOCATE (35,90,70,90)
8330 FRAME
8340 MOVE (40,85)
8350 CSIZE (1,2,90)
8360 PRINT ≡0:≠INSTRUCTION DECODER≠
8370 REM                                                    ** OUTPUT
8380 LOCATE (95,105,10,90)
8390 FRAME
8400 MOVE (95,5)
8410 CSIZE (1,2,0)
8420 PRINT ≡0:≠OUTPUT≠
8430 REM                                                    ** MEMORY
8440 LOCATE (116,197,13,90)
8450 FRAME
8460 MOVE (125,90)
8470 CSIZE (1,2,90)
8480 PRINT ≡0:≠M E M O R Y   C E L L S≠
8490 MOVE (110,85)
8500 CSIZE (1,2,0)
8510 PRINT ≡0:≠CELL CON-   CELL  CON-   CELL CON-   CELL CON-≠
8520 PRINT ≡0:≠NO.   TENT  NO.   TENT   NO.  TENT   NO.  TENT≠
8530 FOR I=1 TO 15
8540 PRINT MOVCS(6+I,45):I≥ NEXT I
8550 FOR I=16 TO 30
8560 PRINT MOVCS(I-9,53):I≥ NEXT I
8570 FOR I=31 TO 45
8580 PRINT MOVCS(I-24,62):I≥ NEXT I
8590 FOR I=46 TO 60
8600 PRINT MOVCS(I-39,71):I≥ NEXT I
8750 MOVE (18,55)
8760 CSIZE (1,1,0)
8770 PRINT ≡0:≠INSTRUCTION≠
```

```
8780 PRINT ≡0:≠    SET≠
8790 PRINT ≡0:≠0-INP≠
8800 PRINT ≡0:≠1-CLA≠
8810 PRINT ≡0:≠2-ADD≠
8820 PRINT ≡0:≠3-TAC≠
8830 PRINT ≡0:≠4-SHF≠
8840 PRINT ≡0:≠5-OUT≠
8850 PRINT ≡0:≠6-STP≠
8860 PRINT ≡0:≠7-SUB≠
8870 PRINT ≡0:≠8-JMP≠
8880 PRINT ≡0:≠9-HRS≠
8890 MOVE (40,67)
8900 CSIZE (1,1,0)
8910 FOR I=1 TO 18
8920 PRINT ≡0:≠¬≠
8930 NEXT I
8940 MOVE (85,68)
8950 FOR I=1 TO 16
8960 PRINT ≡0:≠V≠
8970 NEXT I
8980 MOVE (50,13)
8990 PRINT ≡0:≠<<<<<≠
9000 MOVE (40,90)
9010 CSIZE (6,4,0)
9020 PRINT ≡0:≠A COMPUTER SIMULATOR≠
9030 RETURN
9090 REM
9091 REM                    *********************************************
9092 REM                    *               RESTART MODULE              *
9093 REM                    *********************************************
9094 REM
9100 REM
9110 PRINT MOVCS(22,44);CHR$(27):≠^DDPROGRAM ENDED SUCCESSFULLY≠
9112 GOSUB 9200
9120 GCLR > PRINT CHR$(27):≠H≠> PRINT CHR$(27):≠J≠
9130 PRINT MOVCS(7,10):≠PROGRAM CHOICES ARE:≠
9132 PRINT MOVCS(10,15):≠1-SAMPLE +,-,X,PROGRAMS≠
9134 PRINT MOVCS(11,15):≠2-SAMPLE OUTPUT 1 TO 10 PROGRAM≠
9136 PRINT MOVCS(12,15):≠3-LOAD YOUR OWN PROGRAM≠
9138 PRINT MOVCS(13,15):≠4-STOP ≠
9140 PRINT MOVCS(15,10):
9142 INPUT ≠INPUT YOUR CHOICE: ≠,X2
9144 IF X2=4 THEN STOP
9146 ON X2 GOTO 290,290,290
9148 PRINT MOVCS(23,0):≠WRONG NUMBER, TRY AGAINv≠
9149 PRINT MOVCS(15,29):≠        ≠
9150 GOTO 9140
9190 REM
9191 REM                    *********************************************
9192 REM                    *  CONTINUE AND ERROR MESSAGE ERASE MODULE  *
9193 REM                    *********************************************
9194 REM
9200 PRINT MOVCS(23,44):≠HIT RETURN TO CONTINUE≠
9205 GETKBD ON
9210 IF GETKBD(X)=0 THEN 9210
9220 IF X<>239 THEN 9210
9225 GETKBD OFF
9230 PRINT MOVCS(22,43):≠                                   ≠
9240 PRINT MOVCS(23,43):≠                                   ≠
9250 RETURN
>EXIT
```

R E F E R E N C E S

(CON, 1980) Control Data Education Company, 1980 Catalog. Plato Courses.

(HEW, 1979) Hewlett-Packard, Terminal Basic Manual No. 02647-90005, 1979.

(OME 1, 1970) Ontario Ministry of Education, Curriculum: Elements of Computer Technology, 1970 Senior Division.

(OME 2, 1970) Ontario Ministry of Education, Curriculum: Computer Science, 1970 Senior Division.

(YOU, 1975) Yourdon, E. Techniques of Program Structure and Design. New Jersey: Prentice Hall Inc., 1975.

BIBLIOGRAPHY

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

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

Creative Computing, July 1980, Vol. 6, No. 7.

Kanaroglou, P.S. "A Simulator and a Linker Loader for the Machine Mix", M.Sc. Project, McMaster University, 1979.

Mano, M.M. Computer System Architecture New Jersey: Prentice-Hall Inc., 1976.

Struble, G.W. Assembler Language Programming: The I.B.M. System 360 and 370, Addison-Wesley, Second Edition, 1975.

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