ACROSS COMPILER AND PROGRAMMING SUPPORT SYSTEM FOR THE HP41CV -- ETC(U)

SEP 81  J. N. RICHTMANN
THESIS

A CROSS COMPILER AND PROGRAMMING SUPPORT SYSTEM FOR THE HP41CV CALCULATOR

by

James Norman Richmann

September 1981

Thesis Advisors:  S. H. Parry  R. H. Shudde

Approved for public release; distribution unlimited.
# A Cross Compiler and Programming Support System for the HP41CV Calculator

## Title
Master's Thesis; September 1981

## Author
James Norman Richmann

## Performing Organization Name and Address
Naval Postgraduate School, Monterey, California 93940

## Report Date
September 1981

## Number of Pages
242

## Distribution Statement (of this report)
Approved for public release; distribution unlimited.

## Distribution Statement (of the abstract entered in Block 20, if different from Report)

## Supplementary Notes

## Key Words
Calculator, Cross Compiler, HP41CV Programmable Calculator, Optical Bar Code

## Abstract
With growing Army-wide use of programmable calculators, a system is needed to support the programming and testing of calculator software. This thesis provides a FORTRAN IV program to enable an operations research analyst to more efficiently write and document HP41CV calculator programs. Optical bar code readable by the HP41CV is generated by the program. Also given is an
IBM EXEC II program which provides an interactive programming environment including on-line, self contained instructions. To illustrate the use of the system and the quality of the finished bar code and calculator program listings, examples are given including single variable statistics and linear programming. A final example provides a set of short utility routines which illustrate how programs can be developed for use in a calculator read-only-memory.
A Cross Compiler and Programming Support System for the HP41CV Calculator

by

James Norman Richmann
Captain, United States Army
B.S., Iowa State University, 1971

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September, 1981

Author:

Approved by:

Chairman, Department of Operations Research

Dean of Information and Policy Sciences

3
ABSTRACT

With growing Army-wide use of programmable calculators, a system is needed to support the programming and testing of calculator software. This thesis provides a Fortran IV program to enable an operations research analyst to more efficiently write and document HP41CV calculator programs. Optical bar code readable by the HP41CV is generated by the program. Also given is an IBM EXEC II program which provides an interactive programming environment including online, self contained instructions. To illustrate the use of the system and the quality of the finished bar code and calculator program listings, examples are given including single variable statistics and linear programming. A final example provides a set of short utility routines which illustrate how programs can be developed for use in a calculator read-only-memory.
TABLE OF CONTENTS

I. INTRODUCTION 8

II. THE PROGRAMMING ENVIRONMENT 23

A. CHAPTER OVERVIEW 23

B. STRUCTURED PROGRAMMING WITH THE HP41CV 24
   1. The Need for Structure 24
   2. Fundamental Limitations of Calculators 25
   3. Modular Design 27
   4. Control of Program Flow 28
   5. Clarification of Program Structure 29
   6. Data Types and Indirect Addressing 31

C. ADDITIONAL CRITERIA FOR PROGRAM EVALUATION 33
   1. User Friendliness 35
   2. Execution Speed 37

D. A PROGRAMMING SUPPORT SYSTEM 39
   1. A Cross Compiler and Bar Code Generator 40
   2. A Calculator Emulator 41
   3. A Higher Level Language Compiler 42

APPENDIX A: SINGLE VARIABLE STATISTICS EXAMPLE 44

APPENDIX B: LINEAR PROGRAMMING EXAMPLE 72

APPENDIX C: SUBROUTINES FOR READ ONLY MEMORY 107

APPENDIX D: THE CROSS COMPILER PROGRAM AND COMMAND PROCESSOR 128
LIST OF FIGURES

1. Programming Environment Command Menu --- 12
2. List of Commands --- 13
3. On-Line Introductory Material --- 15
4. Components of Program Documentation --- 21
5. Example Program to Add n Numbers --- 30
6. Program to Recall an Element of a Matrix --- 34
I. INTRODUCTION

For the Army to fight effectively in a resource scarce environment, the quantitative decision making techniques of operations research are important skills for Army staff officers. Staff officers are expected to be able to put numbers in their estimates when briefing commanders. They are expected to be able to measure and evaluate complex operations and subordinate units. They are expected to be frugal managers of time and money. And above all, staff officers must be able to apply sound, quantified reasoning in planning how to win the air-land battle.

The use of hand-held programmable calculators by Army staff officers has the potential for improving the use of quantitative decision making techniques throughout the Army. Faster and more accurate than paper and pencil, the calculator is less expensive and more portable than larger computers. Even when compared to the latest micro-computer systems or to portable terminals used for distributed data processing, the hand-held programmable calculator offers advantages in cost, reliability, power consumption and emission of electromagnetic radiation. Hand-held programmable
calculators have already been successfully used by soldiers in the field for applications in artillery fire direction, surveying, and navigation. In addition, large numbers of Army officers own their own pocket calculators and routinely use them for staff planning and reporting functions.

In January of 1981 the U. S. Army Command and General Staff College at Fort Leavenworth, Kansas selected a programmable calculator for the Combined Arms and Services Staff School (CAS 3.). Using both resident and non-resident instruction, this course is designed to teach all Army captains staff techniques and procedures. As a significant part of the curriculum, the students are introduced to subjects such as statistics and regression, decision theory, combat modeling and linear programming. Considering the large number of officers projected to attend this course in future years, this course represents the most widespread training in operations research techniques ever attempted by the Army. The decision to provide a sophisticated calculator to these students on an experimental basis was made for two fundamental reasons. First, the availability of a calculator with immediate field utility should motivate the student to apply the quantitative techniques as compared to the student who would be forced to do all calculations by
hand. Second, the power of the calculator permits classroom
discussion of techniques such as linear programming and
regression which are very difficult and time consuming to
perform manually.

This thesis documents the author’s work to support the
use of a calculator in the Combined Arms and Services Staff
School. Initially, the intent was to produce a series of
lesson materials incorporating the use of the calculator on
a series of operations research topics which have immediate
application for the Army division level staff officer.
Instead, the work accomplished focused on the design and
construction of a system to make the programming and testing
of calculator programs easier and more efficient. Except
for the introduction, this thesis is written for the person
wishing to implement the programming support system
described. The implementor must have a detailed knowledge
of the instruction set and programming characteristics of
the HP41CV calculator as described in Wickes [Ref. 1: pp.
6-20]. For the eventual user of the system, as compared to
the implementor, the system itself provides on-line document-
tation on how to use the system and what commands and
options are available. Figure 1 shows the command menu dis-
played on the terminal screen by this interactive program:

10
Figure 2 gives a more detailed explanation of each of the commands; and Figure 3 displays the on-line introductory material that is provided to new users of the system. For the user, a knowledge of the information contained in the calculator owner's handbook [Ref. 2] is sufficient to begin writing calculator programs using the support system described.

The calculator selected by the Command and General Staff College, the Hewlett-Packard HP41CV, typifies the state of the art in off-the-shelf calculator technology. While not without disadvantages, this calculator was selected because of its power and features which make it easier for Army staff officers to use. First and most important of these features is the ability of the calculator to manipulate alphabetic characters in addition to numeric data. The calculator can display the name of a variable when input data is required or label output when the calculation is completed. With this feature, the calculator helps the user know what data to input or what action to take next. It also helps alleviate the need for constant reference to printed instructions which are difficult to use under field conditions.
HP41C CROSS COMPILER .......... PROGRAM NAME ............ EDITION=17 SEP 81
SELECT DESIRED COMMAND FROM THE FOLLOWING:

<table>
<thead>
<tr>
<th>PP-KEY</th>
<th>COMMAND</th>
<th>CODE</th>
<th>ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP13</td>
<td>STOP</td>
<td>S</td>
<td>GETS YOU OUT OF THE HP41C CROSS COMPILER</td>
</tr>
<tr>
<td>PP14</td>
<td>HELP</td>
<td>H</td>
<td>SHORT EXPLANATION OF HOW TO USE THE CROSS COMPILER</td>
</tr>
<tr>
<td>PP15</td>
<td>ENTER</td>
<td>E</td>
<td>INTERACTIVE PROGRAM ENTRY (NO FILE CREATED)</td>
</tr>
<tr>
<td>PP16</td>
<td>BAR</td>
<td>B</td>
<td>SUBMIT JOB FOR PHYSICAL PRODUCTION OF BAR CODE</td>
</tr>
<tr>
<td>PP17</td>
<td>NEW</td>
<td>N</td>
<td>BEGIN WORK ON A NEW PROGRAM OR NAMED SUBROUTINE</td>
</tr>
<tr>
<td>PP18</td>
<td>DIREC</td>
<td>D</td>
<td>DIRECTORY OF COMMANDS</td>
</tr>
<tr>
<td>PP19</td>
<td>LIST</td>
<td>L</td>
<td>DISPLAY NAMES OF HP41C PROGRAMS ON DISK</td>
</tr>
<tr>
<td>PP20</td>
<td>OCOMP</td>
<td>O</td>
<td>OFFLINE COMPILE AND AUTO GENERATION OF BAR CODE</td>
</tr>
<tr>
<td>PP21</td>
<td>PRINT</td>
<td>P</td>
<td>PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM</td>
</tr>
<tr>
<td>PP22</td>
<td>*</td>
<td>*</td>
<td>RESERVED FOR FUTURE USE BY HP41 EMULATOR</td>
</tr>
<tr>
<td>PP23</td>
<td>COMP</td>
<td>C</td>
<td>COMPILIE A SOURCE LISTING ON CMS DISK</td>
</tr>
<tr>
<td>PP24</td>
<td>XEDIT</td>
<td>X</td>
<td>EDIT THE PROGRAM USING THE CMS FULL-SCREEN EDITOR</td>
</tr>
<tr>
<td></td>
<td>ERASE</td>
<td></td>
<td>ERASE THE SOURCE FILE, LISTING FILE AND TEXT FILE</td>
</tr>
<tr>
<td>CP</td>
<td></td>
<td></td>
<td>ALLOWS EXECUTION OF ANY VALID CMS COMMAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALLOWS EXECUTION OF ANY VALID CP COMMAND</td>
</tr>
</tbody>
</table>

Figure 1: Programming Environment Command Menu
<table>
<thead>
<tr>
<th>PF-KEY CMD</th>
<th>CODE</th>
<th>ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP13</td>
<td>STOP</td>
<td>THIS COMMAND IS USED WHEN YOU WISH TO STOP PROCESSING HP41C PROGRAMS AND RETURN TO CMS. IF YOU ARE EXECUTING A FUNCTION THAT WAS INVOKED FROM THE COMMAND MENU, IN MOST CASES PP13 WILL RETURN YOU TO THE MENU, AND BY PRESSING PP13 AGAIN YOU WILL RETURN TO CMS.</td>
</tr>
<tr>
<td>PP14</td>
<td>HELP</td>
<td>THIS COMMAND IS USED TO DISPLAY THE DETAILED EXPLANATION OF THE MENU COMMAND PROCESSOR AND ITS AVAILABLE COMMANDS. IF YOU HAVE QUESTIONS ABOUT THE PROCESS OF WRITING ACTUAL HP41C PROGRAMS YOU SHOULD CONSULT THE HP41 OWNER'S HANDBOOK.</td>
</tr>
<tr>
<td>PP15</td>
<td>ENTER</td>
<td>THIS COMMAND IS USED TO ENTER A PROGRAM USING THE CROSS-COMPILER IN AN INTERACTIVE MODE. THE ADVANTAGE OF THIS MODE IS THAT ANY SYNTACTICAL ERRORS IN THE HP41C PROGRAM ARE IMMEDIATELY IDENTIFIED BY THE CROSS-COMPILER AND AN ERROR MESSAGE IS SHOWN ON THE SCREEN. THE DISADVANTAGE IS THAT THE USER IS TOTALY RESPONSIBLE FOR UPPER AND LOWER CASE BEING ENTERED PROPERLY.</td>
</tr>
<tr>
<td>PP16</td>
<td>BAR</td>
<td>THIS COMMAND IS USED ONCE THE HP41C PROGRAM IS WRITTEN AND COMPILED WITHOUT ERRORS. IT SUBMITS A JOB TO MV'S BATCH FOR THE PHYSICAL PRODUCTION OF THE BAR CODE.</td>
</tr>
<tr>
<td>PP17</td>
<td>NEW</td>
<td>THIS COMMAND IS USED TO DIRECT THE ATTENTION OF THE COMMAND PROCESSOR TO A NEW HP41C PROGRAM SOURCE FILE. WHEN USED TO INIITATE NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH FILETYPE &quot;HP41&quot; AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.</td>
</tr>
<tr>
<td>PP18</td>
<td>DIREC</td>
<td>THIS COMMAND DISPLAYS THE FULL COMMAND MENU. IT HAS PRIMARY USE WHEN YOU FINISH AN OPERATION THAT FILLS THE SCREEN WITH TEXTUAL MATTER AND YOU RECEIVE ONLY THE PROMPT &quot;INPUT COMMAND&quot;.</td>
</tr>
</tbody>
</table>

Figure 2: List of Commands
<table>
<thead>
<tr>
<th>PF-KEY CMD</th>
<th>CODE</th>
<th>ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF19 LIST</td>
<td>L</td>
<td>THIS COMMAND DISPLAYS &quot;FLIST&quot; FOR THOSE HP41C PROGRAMS THAT ARE ACTIVE ON YOUR A DISK. FROM THIS LIST, YOU CAN ERASE OLD PROGRAMS TO RELEASE DISK STORAGE, CHANGE THE NAME OF PROGRAMS, OR EXAMINE THE CONTENTS OF ANY PROGRAM.</td>
</tr>
<tr>
<td>PF20 OCOMP</td>
<td>C</td>
<td>THIS COMMAND IS USED TO PRODUCE AN &quot;OFFLINE&quot; COMPILATION OF THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPILATION WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.</td>
</tr>
<tr>
<td>PF21 PRINT P</td>
<td>P</td>
<td>THIS COMMAND PRINTS A COPY OF THE &quot;LISTING&quot; FILE ON THE HIGH SPEED PRINTER. IF YOU WISH TO HAVE A PRINTED COPY OF THE SOURCE CODE WITHOUT THE CROSS-COMPILER'S FEEDBACK, IT IS BEST TO SIMPLY PRINT THE SOURCE CODE CMS FILE BY ISSUING THE CMS PRINT COMMAND.</td>
</tr>
<tr>
<td>PF22 GO</td>
<td>G</td>
<td>THIS COMMAND IS USED TO INVOKE THE HP41C EMULATOR PROGRAM WHICH ALLOWS YOU TO TEST EXECUTION OF THE PROGRAM ON THE LARGE COMPUTER. THE EMULATION PROGRAM WILL EXECUTE THE PROGRAM EXACTLY AS YOUR CALCULATOR WOULD. THIS COMMAND HAS NOT BEEN IMPLEMENTED AS OF 17 SEP 81.</td>
</tr>
<tr>
<td>PF23 COMPC</td>
<td>C</td>
<td>THIS COMMAND IS USED TO INVOKE THE CROSS COMPILER TO TRANSLATE AN HP41C PROGRAM WRITTEN ON CMS DISK IN SOURCE CODE FORM. AFTER THE COMPILATION THE USER IS AUTOMATICALLY PLACED IN THE CMS BROWSE MODE FOR THE OUTPUT &quot;LISTING&quot; FILE THAT RESULTED FROM THE COMPILATION.</td>
</tr>
<tr>
<td>PF24 XEDIT X</td>
<td>X</td>
<td>THIS COMMAND IS USED TO INVOKE THE FULL-SCREEN EDITOR TO MAKE MODIFICATIONS TO THE HP41C SOURCE CODE FILE.</td>
</tr>
</tbody>
</table>

Figure 2 (Continued)
HP41C CROSS COMPILER COMMAND PROCESSOR

YOU ARE CURRENTLY EXECUTING A CMS EXEC FILE THAT MAKES IT EASY TO INVOKE
THE HP41C CROSS COMPILER AND WRITE PROGRAMS USING CMS AND THE IBM 3278
DISPLAY TERMINAL. COMMON PROGRAMMING REQUIREMENTS SUCH AS EDITING CAN
BE ACCOMPLISHED IN THREE WAYS:

--USING THE PROGRAMMED FUNCTION KEYS (PF KEYS)
--USING A SHORT COMMAND WORD
--USING A ONE OR TWO LETTER MNEMONIC CODE

THE COMMAND ACTIONS AND THEIR ASSOCIATED PF KEYS AND CODES ARE ALL GIVEN
IN A DIRECTORY WHICH IS DISPLAYED WHEN THE COMMAND PROCESSOR IS WAITING
FOR YOUR INPUT.

IN ORDER TO GO FROM A PROGRAM IN YOUR HEAD TO THE FINISHED BAR CODE
THERE ARE THREE MAIN STEPS:

(1) EDIT. THE PROGRAM MUST BE PREPARED AS INPUT TO THE CROSS
COMPILER. THE EASIEST WAY TO DO THIS IS WITH THE
CMS XEDIT FACILITY.

(2) COMPILE. THE PROGRAM MUST BE PROCESSED BY THE CROSS-COMPILER.
THE CROSS-COMPILER IS ACTUALLY A FORTRAN PROGRAM
WHICH PRODUCES TWO CMS FILES AS OUTPUT. BOTH
THOSE FILES HAVE THE SAME NAME AS YOUR PROGRAM NAME,
BUT HAVE DIFFERENT FILE TYPES. THE "LISTING" FILE
SHOWS THE RESULTS OF THE COMPILER STEP INCLUDING ANY
ERRORS, AND THE "DATA" FILE IS A FILE OF ZEROS AND
ONE'S USED BY THE BAR CODE GENERATOR.

(3) BAR. THE "DATA" FILE FROM THE COMPILER STEP IS USED AS INPUT
TO PRODUCE THE ACTUAL BAR CODE. YOU SHOULD NEVER PER-
FORM THIS STEP UNLESS YOUR PROGRAM HAS SUCCESSFULLY
COMPILED WITHOUT ERRORS. THIS STEP IS DONE BY THE
BATCH PROCESSOR AND IT MAY TAKE SEVERAL HOURS TO GET
YOUR FINISHED BAR CODE.

Figure 3: On-Line Introductory Material
A second important feature is the multiplicity of means by which programs can be entered into the calculator. Magnetic cards, read only memory, and optical bar code are all available and each has advantages depending on the situation. For the long term, read only memory offers the ability to retain very large programs (in excess of 8000 bytes) and the simplest and most reliable means of entering programs into the calculator under field conditions. For the short term, optical bar code offers the least expensive method of reproducing and distributing calculator software that has not been subject to extensive field testing. In addition, as shown in this thesis, the optical bar code can provide an important link between a main-frame computer and the hand-held calculator.

A third important feature of the HP41CV is its relatively large memory capacity as compared to programmable calculators such as the Texas Instruments TI-59. A large amount of memory permits the solution of larger, often more realistic problems than could previously be solved on a hand-held device. A demonstration program given in this thesis for linear programming is an example of an application where the full memory capability of the HP41CV is required to be able to solve realistic problems.
To take advantage of the calculator's unequalled ease and portability, the operations research analyst is challenged to overcome its limits of speed and memory capacity. The preparation of calculator software is as difficult, if not more so, than the preparation of software for larger computers. To accomplish the most possible with the hand-held device, the calculator programmer is often forced to write programs which are very difficult to comprehend when examined by other programmers. As Dahl, Dijkstra and Hoare (Ref. 3: pp. 1-10) point out there are limits to human competence which interfere with the programming process. In the past, with less mature calculators which constrained the typical program to a few hundred program steps, these limits to human competence were neither as apparent nor as economically important as they are with the HP41CV. Accordingly, it is not envisioned that the average Army officer who uses the HP41CV on real world problems which push the calculator to the limits of its capability would write their own programs. In particular, it was never intended that the students in the Combined Arms and Services Staff School would be taught calculator programming. It is a tribute to the power of the device and the quality of the calculator software when a relatively inexperienced user can run complex...
programs using little more than the digit entry keys and the run-stop key on the calculator. This does not mean that the user must not have a clear understanding of his problem or the solution technique, but rather it means that the calculator should not require programming skill or extensive training prior to application.

The growing complexity of calculator programs described above and the realization that calculator programs for Army field use are not programmed in the field, suggest the need for a system to support the development, distribution and maintenance of calculator software. An operations research analyst or other professional programmer must be able to more efficiently prepare calculator programs than by keying them into the hand-held device. By preparing the programs initially on a larger computer, such as the IBM 3033, the programmer can use the speed and storage capability of the larger machine to great advantage. In addition, the availability of a full-screen video text editor speeds the process of program revision and maintenance. By providing a capability to integrate comments directly into the source code on the larger computer, program documentation is more easily provided. Essentially the idea is that a programmer would write the calculator program using a terminal
connected to a large computer. After the calculator program is entered into the large computer, a **compiler** program running on the large computer would check the calculator program for errors and convert the mnemonic instructions into the "key codes" which are the numeric instructions actually executed by the calculator. Then an **emulator** program running on the large computer would take the numeric instructions from the compiler and execute the program—in effect making the large computer produce the same effects as the calculator only much faster and more efficiently for the programmer. Finally, when the program has been written and tested on the large computer, optical bar code is produced which allows for the economical distribution and use of the program in the field. To encourage the calculator programmer to use the system described, this process should occur in an interactive programming environment in which the user can move from one step to another by issuing simple commands such as those listed and described in Figure 2 and receive help or on line instruction whenever desired. Under this proposed system, the advantages of both the larger computer and the hand-held calculator are used appropriately in a mutually supporting manner. This thesis presents two of the components of this proposed system. First, an IBM EXEC II
program is given which provides an interactive programming environment for users operating under IBM's Conversational Monitor System (CMS). A short discussion of the design of this program and a complete copy of the source code is contained in Appendix D to this thesis. Secondly, a cross compiler written in IBM standard FORTRAN IV is provided for translating calculator mnemonic instructions into the key codes necessary for use by the emulator and also for the production of optical bar code. The term cross compiler refers to the fact that the program runs on one machine (the larger computer) but compiles programs for another machine (the calculator). A discussion of the design of this program and a complete copy of the source code is contained in Appendix D. To make the program easier to understand and adapt to new requirements, it is modularized into 24 subroutines and is heavily commented.

To illustrate the use of the system, two of the six example programs originally planned are provided in this thesis. Revised plans now call for the remaining four example programs to be issued at a later date as Naval Postgraduate School technical reports. Because the reasons for the delay constitute some of the most important lessons learned from this thesis research, Chapter 2 documents the process
with a technical discussion of the factors involved. The major conclusions described in Chapter 2 are the need for a prioritized list of criteria with which to evaluate calculator programs and the need for more structure in the programming process. Chapter 2 is technically oriented and assumes the reader is familiar with the concepts of structured programming.

Each of the calculator program examples is described in a separate appendix in which the documentation listed in

1. Program Description
2. Sample Problem
4. Source Code Listing with Comments
5. Bar Code

Figure 4: Components of Program Documentation

Figure 4 is provided. The first example on single variable statistics is documented in Appendix A and uses the calculator in an area where calculators have long been used, but does so in a way that shows the unique capabilities of the
A second example on linear programming is documented in Appendix B and illustrates an area where calculators have not received widespread application. Most calculator linear programs which have been published to date have been either incomplete algorithms or have been limited to very simple problems.

A third example, which by its nature does not conform to the documentation standards outlined above, describes a set of utility routines which could be distributed in read only memory. Programs for read only memory have different characteristics from other calculator programs and Appendix C is provided to illustrate some of these differences.
II. THE PROGRAMMING ENVIRONMENT

A. CHAPTER OVERVIEW

This Chapter examines calculator programming within the context of the author's experience in preparing HP41CV programs in support of the Combined Arms and Services Staff School. With the advanced capabilities and features of the HP41CV, it was hoped that a complete package of software could be prepared quickly. To document why this did not occur, this chapter will examine strengths and weaknesses of the calculator in relationship to a collection of techniques referred to in computer science as structured programming.

For the reader unfamiliar with this term, the previously cited work by Dahl, Dijkstra, and Hoare [Ref. 3] is recommended. This chapter is technically oriented and does assume familiarity with structured programming concepts.

When programming calculator programs for personal use, most programmers, including the author, do not find the task difficult. Programming a hand-held calculator with the capabilities and features of the HP41CV can be a rewarding experience. It is rewarding to master the algorithms of an operations research technique on a hand-held device. The
educational value in programming the calculator has been recognized by many educators, including Hamming [Ref. 4: pp. 2-3] and Weir [Ref. 5: pp. xii-xiii]. Providing a program for general distribution which makes optimum use of the calculator is quite a different situation. It was the author's experience that programs, which gave correct answers when used by the author, often had to be completely re-written several times before being acceptable. This problem became more acute as the size of the programs grew beyond 400 program steps, for at that size it became increasingly difficult to modify programs without affecting the total design. The major conclusions described in this chapter are the need for a prioritized list of criteria with which to evaluate calculator programs and the need for more structure in the programming process.

B. STRUCTURED PROGRAMMING WITH THE HP41CV

1. The Need for Structure

To increase the efficiency of the programming process, a collection of techniques known as structured programming has received widespread attention in the computer science community. While there is no one definition of structured programming, it does require three essential characteristics. First, there must be a logical structure
to the program which reflects the nature of the problem to be solved and any constraints imposed upon the solution. Second, the systematic process of stepwise refinement is used to limit the complexity of program segments. Third, the programming language must reflect the logical structure of the program and assist in stepwise refinement. These three characteristics represent not so much a detailed recipe for program development as they do a philosophy of how programs can be more efficiently written. It was with this philosophy in mind, that a calculator programming support system was proposed which could take into account the strengths and weaknesses of the calculator; balance the structured programming philosophy with the other criteria listed below; and thereby solve the problems encountered in writing calculator software for the Combined Arms and Services Staff School.

2. **Fundamental Limitations of Calculators**

Writing programs to solve complex problems on a hand-held calculator is difficult both because of inherent limitations in the calculation speed and memory capacity of the machine and also the inability of the calculator's native programming language to directly support structured programming constructs. In many respects, the task is
similar to writing assembly level language programs for larger computers. Calculator programming features a powerful instruction set including advanced mathematical functions but lacks any ability to refer to variables by name instead of storage address. Like assembly language, the calculator's programming language consists of short mnemonic instructions typically followed by the storage location of the data to which the operation is to be applied. While a large amount of computer programming is still done in assembly language, it is generally accepted that programming in a higher level language such as FORTRAN is preferable. Programs written in an assembly language take more time to write and are not as easily changed as higher level language programs. Also, because they depend on the instruction set of a particular machine, they can not be easily transferred from one computer to another. These same disadvantages apply to calculator programming. In addition, because the hand-held device does not have the speed and memory capability of the larger machine, the calculator programmer must be even more mindful of the need to optimize his program to save program steps and execution time.
3. **Modular Design**

The HP41CV supports structured programming as well or better than any other hand-held calculator. As described in the owner's manual [Ref. 2: pp. 177-196], the machine primitive instruction XEQ encourages the construction of modular programs using calculator subroutines. Each subroutine can be a self-contained unit capable of being written and tested independently and used by multiple programs. This modularity is most strongly encouraged when routines in read only memory are used, for then the application programmer can significantly reduce the number of program steps in his own program. This modularity, however, is not complete, since all variables are globally referenced and can be changed deliberately or inadvertently by any subroutine.

This problem is no more apparent than with the use of read only memory, since one of the most limiting factors in using the read only memory programs as subroutines is conflict in the use of common registers. Also, unlike the modularity required in truly structured programs, there is no restriction limiting a subroutine to a single entry and a single exit point. In structured programs, such limits on entry and exit serve to define the fundamental building blocks by which stepwise refinement is made possible. With
the calculator, however, multiple entry and exit points are most useful for allowing a common routine to handle a duplicity of problem conditions. In this thesis, for example, programs are given for which two standard entry points are provided. One entry point uses an alpha-numeric label and an audio prompt to speed data entry, while a second entry point uses the alpha-numeric label but suppresses the audio tone. After data entry, the value entered is displayed, and the user is required to verify the accuracy of the data entered. By using the same subroutine with different entry points, memory space is saved overall at the sacrifice of the structured programming philosophy.

4. Control of Program Flow

A basic deficiency prohibiting the HP41CV from directly supporting structured programming is the way in which program flow is controlled. Programming languages which support structured programming typically have instruction constructs such as WHILE--ENDWHILE, REPEAT--UNTIL, or LOOP--QUIT--ENDLOOP which make programming loops clean and concise. Constructs such as IF--THEN--ELSEIF--ELSE--ENDIF and the CASE statement make the evaluation of conditional expressions efficient and relatively error free. Also, structured programming languages typically discourage the
use of GOTO unconditional transfers because they lead to confusing code. In contrast, the HP41CV programmer must write his own looping constructs and his own conditional evaluation constructs using machine primitive instructions which somewhat obscure the program's basic objective and flow of control. In addition, it is difficult to avoid disturbing pending operations in the stack registers when a conditional statement must be evaluated. As can be seen by the short program shown in Figure 1, the notation of the programming language does not permit structured program flow.

5. Clarification of Program Structure

Because no calculator, including the HP41CV, supports named variables, the use of comments as an integral part of the calculator program is vital if the logical structure of the program is to be made clear as required by structured programming. Comments should provide the variable names when storing and recalling data; they should provide clarification of program flow; and they should mark subroutine boundaries and entry and exit points to make it easier to identify segments of the program. With the HP41CV's stack oriented architecture, it is also frequently useful to display the names of the contents of each of the
Given the number \( n \) in the \( x \)-register, this program fragment will sum the data values stored in memory locations 1 through \( n \).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBL &quot;SUM&quot;</td>
<td>To execute press &quot;XEQ SUM&quot;.</td>
</tr>
<tr>
<td>1E3 / 1 + 0</td>
<td>Establishes a loop counter.</td>
</tr>
<tr>
<td>RCL IND Y</td>
<td>Recall the next data value.</td>
</tr>
<tr>
<td>+</td>
<td>Accumulate the sum.</td>
</tr>
<tr>
<td>ISG Y</td>
<td>Increment the loop counter.</td>
</tr>
<tr>
<td>GTO 00</td>
<td>If more data remains, branch;</td>
</tr>
<tr>
<td>RTN</td>
<td>else, quit and display sum.</td>
</tr>
</tbody>
</table>

Figure 5: Example Program to Add \( n \) Numbers
stack registers. In Appendix C on common subroutines with read only memory application, a shell sort [Ref. 6: pp. 84-95] routine is given which employs the technique of using comments to display the names of the variables on the stack register.

6. Data Types and Indirect Addressing

Calculator programs represent more than a sequence of keystrokes; they also represent the manipulation and transformation of data. For maximum efficiency, the manipulation of data should be structured so as to prevent common programming errors. For this reason, most computer languages which directly support structured programming enforce data type correspondence between data and operations. Frequently the formal declaration and initialization of variables is also required. The HP41CV handles two types of data—real numbers and alphanumeric characters. While no formal declaration of variables is required, type checking is done automatically and is transparent to the user. Any attempt to perform an arithmetic operation on alpha-numeric data will result in the message "ALPHA DATA" and the program will halt.

Because there is no formal declaration of variables, the programmer writing programs for the HP41CV must use
extreme caution in managing his data set and insuring that
the numbers stored and recalled by the calculator program
are in fact the data elements desired. A typical example of
an improper data reference occurs when a program is using
indirect addressing and attempts to store or recall data
from a non-existent data register. This programming error
is so common that a special error message "NONEXISTENT" is
provided by the calculator when this error is detected.
Indirect addressing is an important feature which gives the
calculator a considerable amount of power and flexibility,
but also represents an additional responsibility for the
programmer to explicitly control. On the HP41CV all indi-
rect addressing calculations must be specifically provided
by the application program--there are no vector or array
data types such as usually found with higher level lan-
guages. In an attempt to make indirect addressing more
transparent to the programmer, an experimental subroutine
was prepared to recall an arbitrary element of a matrix
stored as a two dimensional array. This subroutine, which
is shown in Figure 2, was used in a simultaneous differen-
tial equation combat model and the results evaluated. It
accomplished the task, but slowed the execution of the pro-
gram considerably (resulting in an overhead of 10.5 seconds
of extra execution time for every 100 subroutine calls) and did not significantly improve the size or legibility of the application program. Accordingly, this technique is not recommended and indirect addressing remains a task that must be treated explicitly by the application programmer.

C. ADDITIONAL CRITERIA FOR PROGRAM EVALUATION

Calculator programming in many respects resembles a multi-criteria decision problem. On the surface the criteria for program effectiveness are quite straightforward—the program must yield the correct answer, run quickly, require the fewest possible memory registers and be user friendly. Unfortunately, these objectives often conflict and can not always be simultaneously achieved. In particular, the principles of structured programming are often in conflict with the desire to reduce the size of programs and increase their execution speed. It is also true that the objectives of structured programming concern the process of writing programs, whereas the additional criteria listed concern the final program product itself and are therefore logically considered separately. Attempting to achieve all criteria at once can lead to failure, and some tradeoffs must be considered to evaluate programs and guide program development. The following criteria represent
Entry to this routine assumes the x register contains the column number and the y register contains the row number. The base address must be stored in R04 and the dimension of the matrix must be stored in R05.

1 LBL "RCLM
2 RCL 04 (BASE ADDRESS REGISTER
3 + (ADD BASE TO COLUMN NUMBER
4 X<>Y (RECALL THE ROW NUMBER
5 1
6 -
7 RCL 05 (DIMENSION OF THE MATRIX
8 *
9 + (ADDRESS IS NOW IN X REG
10 RCL IND X (RECALL THE DATA DESIRED
11 RTN
12 END

Figure 6: Program to Recall an Element of a Matrix
"lessons learned" in developing application programs as examples for this thesis.

1. **User Friendliness**

User friendly programs consider the application environment and do not task the user to be all knowing or without error in entering data. While individuals differ greatly with experience, the average user will make frequent errors in entering data with the hand-held calculator's small keyboard. In talking with officers who had used the TI-59 calculator in the field for fire direction, it was discovered that most preferred to use the printer with the calculator because it allowed data to be checked after entry. This was in spite of the fact that the printer and calculator combination is more costly, less portable and less suitable for use in the field than the calculator alone. In short, user friendliness was more important than these other criteria. For this reason, it should be mandatory that any calculator programs intended for Army use in the field must allow the verification of data after entry. Because the use of the printer obviates many of the advantages of the hand-held calculator, the printer should not be required for this verification. One of the considerable advantages of the HP41CV is that the large amount of program
memory makes it possible to store the input values and perform this verification. However, programs written with this criteria in mind may not appear be most efficient to the casual observer.

Another important aspect of user friendliness is limiting the complexity of the calculator and the actions required to get results. The typical Army officer has little appreciation for the multitude of scientific and mathematical functions labeling the keys of the HP41CV. Yet the common programming practice of using the top two rows of calculator keys to indicate the identity of a variable either upon input or output increases confusion over the use of the function keys. This works as follows: When local alphabetic labels are used in a program to represent entry points by which a user indicates the identity of an input variable or requests a particular output variable, then the first two rows of keys on the HP41CV become subroutine execution keys pointing to these local labels when the calculator is in user mode. This feature was very important on the HP67 and TI-59 where the lack of alpha-numeric capability required this method of program execution in order to most easily determine the identity of the input or output value, but it is less important on the HP41CV. It is almost always
true that a program which requires the use of local labels is harder to use, and requires more frequent reference to the user instructions than a program which uses only the run-stop key and properly prompts the user and labels output values.

2. Execution Speed

The second most important criteria for a calculator program is that it must yield results relatively quickly. In preparing example programs for this thesis, this point became very clear when testing two particular programs. One program, a simultaneous differential equation combat model, required in excess of 150 data values in order to yield results. It should be noted that it was only with the introduction of the HP41CV that it became feasible to consider such large problems on a hand-held device. To accommodate the size of the model, the program was written so as to economize on program steps at the expense of increased execution time. It became immediately obvious upon initial testing that this had been the wrong priority—for users of the program were not impressed with either the use of the calculator or the utility of the combat model. If such user acceptance is not present, then the calculator program will remain unused, no matter how elegant the design to conserve
memory. In contrast, the linear programming example given in Appendix B was written so as to emphasize speed even if it meant including code redundancy. This program has been well received in part because it is so much faster than paper and pencil methods.

The easiest and most effective technique that is useful in increasing speed is to decrease the number of program steps that the calculator must process inside program loops. For example, if two different program options require similar but slightly different actions within a program loop, it is tempting to insert a program flag check and branching instructions within a loop so as to use the same loop for both conditions. But this means that the calculator must test the flag and branch inside the loop even though the program is probably shorter overall.¹

Instead, if the application permits, the memory capacity of the HP41CV can be used to best advantage by testing the flag once and then providing separate program loops for the two conditions. Again, this does not appear elegant to the casual observer, but it may result in a more successful program overall. This principle was discovered while

¹ Branching is required when the flag tests either set or clear if more than one instruction is required to account for the differences in the two conditions.
programming the single variable statistics program given in Appendix A. Initially, this program used a common loop for all data input and output operations, including reviewing the input data and making individual corrections. By providing a separate, somewhat redundant loop for data correction, the time required to input data points was reduced.

D. A PROGRAMMING SUPPORT SYSTEM

Considering the structured programming philosophy discussed above in paragraph B and the additional criteria for evaluating programs listed in paragraph C, it becomes immediately obvious that programming with the calculator alone will never meet even a majority of these objectives. It must be recognized that the problem under consideration is not how the average person who owns a calculator should proceed to program it for his own personal use, but rather how the Army can best provide the most cost-effective computational resource for field use. For these reasons, a comprehensive programming support system is required. The programming support system outlined here will consider only the requirement for cost-effective preparation and maintenance of the calculator programs and not the broader issues of distribution and logistic support for the entire
calculator system to include hardware, training materials and printed references.

1. A Cross Compiler and Bar Code Generator

The first requirement for an operational support package is to free the programmer from the limitations of the hand-held calculator itself. Even with the printer and other peripherals, the calculator is no match for the larger machine when large programs must be examined or edited. In addition, the calculator is not currently capable of producing its own optical bar code as required for economic reproduction and distribution of the software. Accordingly, a cross compiler for the HP41CV was listed as the first requirement of the programming support system. Such a cross compiler has been written and is the major outcome of this thesis effort. This cross compiler accepts an HP41CV program written in the language of the calculator and returns the finished bar code as output. Any valid HP41CV program will be processed without need for modification by the cross compiler. In addition to the basic language of the calculator, the user is allowed to inject comments directly into the source code with the use of the left parenthesis as a comment indicator mark. The ability to make comments directly in the source code makes the calculator programs
more legible and more easily modified at a later date or by another programmer. Often, well placed comments can make up for a lack of structure in the program itself as far as legibility and maintainability are concerned. Having the comments directly in the source code facilitates their use and helps insure that they are as up to date as the program. For the average programmer, use of unmodified HP41CV source code augmented with a comment indicator will represent the most common use of the cross compiler. The cross compiler is described in more detail in Appendix D including a complete listing of the source code.

2. A Calculator Emulator

After the calculator source code has been processed by the cross compiler, a need exists to be able to run the program without the wait for the generation of bar code. In addition, for the future development of read only memories for the calculator, an emulator program is required because the calculator itself can store only up to 2000 instructions in active random access memory. The read only memory can store up to four times this amount. Thus, the calculator by itself may not be capable of testing extremely large programs or programs with large amounts of constant data also stored in the read only memory. Although an emulator was
not written for this thesis, the design of the cross compiler reflects the need for such a program. For example, the cross compiler generates an intermediate array of decimal integers which represent the machine language of the HP41CV prior to conversion to binary. It was intended that these decimal integers could be used without modification or further translation within a FORTRAN computed goto statement. Thus, with the difficult translation, instruction parsing and syntax recognition already performed by the cross compiler routines, the emulator could consist of one large FORTRAN loop wherein a decimal integer was addressed in the instruction array by a program pointer variable. The integer is then immediately sent to a computed goto statement which would branch to the appropriate line of FORTRAN code which would simulate the referenced instruction, including updating the stack and the program pointer as appropriate.

3. **A Higher Level Language Compiler**

The final component in the calculator programming support system would be a program that would translate a higher level language such as PASCAL into HP41CV language which could then be sent to the cross compiler for verification and generation of the bar code and intermediate
calculator language listings. It is the higher level
language compiler that would most directly make up for the
weakness of the calculator in supporting structured program-
ing. It would be able to increase the modularity of pro-
grams, provide for named variables, make indirect addressing
transparent and provide structured statements such as
WHILE--ENDWHILE and IF--THEN--ELSE. Again, the design of
the cross compiler anticipates this requirement and provides
a considerable number of subroutines that would also be
required by a higher level language compiler. These subrou-
tines include a complete set of string functions for manipu-
lating character data in FORTRAN and an instruction parser.
Because it was envisioned that the higher level language
compiler would also be able to process statements entered
directly as HP41CV instructions, the cross compiler is con-
structed so that the routine which compiles individual lines
of HP41CV source code could be called as a subroutine by the
higher level language compiler. Thus, all three major com-
ponents of the proposed calculator programming support
system would work together efficiently.
INTRODUCTION:

Calculating single variable statistics is one of the most frequently used applications of programmable calculators. Army division level staff officers use single variable statistics to summarize and describe data for command briefings and periodic reports. The text by Mendenhall, Scheaffer and Wackerly [Ref. 7: pp. 3-13] is recommended as an introduction to the statistical measures calculated by the program given in this appendix. This program automatically calculates:

- Mean and Median
- Sample Standard Deviation
- Sum of the Squared Deviations about the Mean
- Coefficients of Skewness and Kurtosis
- Minimum, Maximum and Range
- Histogram Cell Frequencies
A single variable statistics program has been given as an example because of its immediate utility to the staff officer and to illustrate several features of the HP41CV which make it a superior device for Army field use. The most important of these features is alphanumeric prompting for input data values. The program given in this appendix provides an alphanumeric prompt for every input and output value and requires only the digit entry keys and run/stop key for data entry. Another important feature of the HP41CV used by this program is its large memory capacity. This program retains up to 219 data points in the calculator's memory to allow the user to review the input data and make corrections during data entry. The large amount of memory allows the calculator to sort the data and calculate the order statistics including the minimum, maximum and median. Calculation of the median is a feature of this program which distinguishes it from other calculator statistics programs. In addition, without having to re-enter the data, the histogram may be calculated with a varying number of cells or a varying cell width.

PROGRAM DESCRIPTION:

The single variable statistics program has entry points for two different techniques of data input. The fastest
method, which provides both an alphanumeric prompt and an audio tone to speed data entry, may be called by execution of the program from entry point "STAT1." A slower method, which provides greater accuracy and suppresses the audio tone for classroom use, may be called by execution of the program from entry point "S1." When called from "S1," the program requires the verification of each data point after entry. The sequence of actions is as follows:

1. The calculator displays an alphanumeric prompt. As an example, "X1?" is the prompt for the first point.
2. The user enters the data value with the digit entry keys and presses the run/stop key.
3. The calculator displays the data entered with a label derived from the alphanumeric prompt. For example, "x1=3.1415" is a typical calculator response. This display is prompting the user to verify the correctness of the data displayed.
4. If the value is correct, then the user simply presses the run/stop key and the calculator advances to the next point.
5. If the value is erroneous, the user enters the correct value with the digit entry keys and then presses the run/stop key. Then the calculator will again repeat step 3 and ask the user to verify the data value. This process will continue until the user makes no modification to the data value.

To run the program from either entry point the user may use the XEQ key, or assign the entry point label ("STAT1" or "S1") to a key and execute it by pressing that key in the USER mode. Further instructions on running programs and making key assignments are contained in the calculator owner's manual [Ref. 2: pp 114-116].
In addition to the two initial entry points described above, several other alphabetic labels provide the user with functions that are called outside the normal sequence of program execution. Label "SR" provides the user with the capability to review the data stored in calculator memory, either before or after the data has been sorted. When used before the sort, the "SR" function is most useful in verifying the entire data set at one time. If used for this purpose, it should be called after all of the data has been entered and the mean of the data set is displayed with the "XBAR" label. If flag 21, the printer enable flag, is set "on" during this data review, then the calculator will stop as each point is displayed and the user may make corrections in the same manner as described above for the point-by-point verification associated with the "SI" entry point. When used after the sort, the "SR" function is most useful for displaying the order statistics for the data set. If used for this purpose, it should be called after the histogram is output--when the "CMD" prompt is displayed. If the user presses run/stop after the "CMD" prompt, the order statistics will automatically be displayed.

The design of the program, especially the data entry loop, reflects the need for calculation speed. Code
redundancy exists at several points in order to reduce the need for extra flags, labels and goto statements which would slow execution during data entry. In spite of this need for speed, the summary totals needed for calculation of mean, standard deviation, skewness and kurtosis are accumulated during data entry. This is done so that these summary statistics are available with little or no wait following data entry.

A complete listing of the program registers and flags used by this program is shown at the end of the program listing.

SAMPLE PROBLEM:

In order to establish a training standard for an obstacle course, a division assistant randomly selects 10 soldiers and records the time it takes each to complete the course. The following times in minutes were recorded:

2.1 2.4 2.2 2.7 2.5
2.4 2.6 2.6 2.3 2.9

Determine the summary statistics and cell frequencies necessary to plot a histogram of this data.
SOLUTION:

1. First, set the size of the calculator's data memory large enough to retain the data values. This requires at least 16 registers plus 1 for each data point, or a total of 26 in this example. Alternatively, the size of data memory may be set arbitrarily large, up to a maximum of 235 provided the user has no other programs in the calculator he wishes to retain. For this example press:

\[ \text{XEQ ALPHA SIZE ALPHA 26} \]

2. To call the program, determine the appropriate method of data entry and select the corresponding entry point. Press:

\[ \text{XEQ ALPHA STAT1 ALPHA (quick entry)} \]
\[ \text{XEQ ALPHA S1 ALPHA (classroom use)} \]

3. The calculator will respond with the prompt "N?" asking for the number of data points. Press:

\[ 10 \text{ R/S} \]

4. The calculator will respond with the prompt "X1?" asking for the first data point. Press:

\[ 2.1 \text{ R/S} \]
5. If you called the program via "S1" the calculator will respond with "X1=2.100" asking for verification that the first point is correct. If not correct enter the correct value, else press run/stop.

6. The calculator will continue in the same way as steps 4 and 5 for the remaining data points until all the data has been entered. If at any time you discover that you have made an error in data entry for any point, press:

XEQ ALPHA SC ALPHA

The calculator will respond with the prompt "POINT?" asking for the number of the point in error. For example, if point number 5 were in error, you would then press:

5 R/S

Assuming you had just input a 5 as the point in error, the calculator would then respond with the prompt "X5?" asking for the correct value of point 5. Respond with the correct value and press run/stop. The calculator will then go back to the place in the data entry sequence where it left off or it will go to the calculation of the summary statistics if data entry was previously completed.
7. When data entry has been completed, the calculator will respond with the mean of the data sample labeled as follows:

XBAR=2.470

At this point, you have the option of reviewing the entire data set or continuing to calculate the remainder of the statistics. To review the entire data set, press:

XEQ ALPHA SR ALPHA

Note that if flag 21 is set on (press SF 21), the calculator will stop after each data point is displayed, permitting you to change any value simply by entering the new value and pressing run/stop.

8. After the mean is displayed with the "XBAR" label, if you simply press the run/stop key, the calculator will calculate the following statistics with the label shown: After each press R/S.

<table>
<thead>
<tr>
<th>Display</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSQD=0.521</td>
<td>Sum of Squared Deviations</td>
</tr>
<tr>
<td>SX=0.241</td>
<td>About the Mean</td>
</tr>
<tr>
<td>SKEW=0.170</td>
<td>Sample Standard Deviation</td>
</tr>
<tr>
<td>KURT=2.302</td>
<td>Skewness</td>
</tr>
<tr>
<td></td>
<td>Kurtosis</td>
</tr>
</tbody>
</table>

9. At this point the calculator will automatically sort the data. This may take from several seconds to several minutes
depending on the number of points in the data set. After the data set has been sorted, the calculator will display the median as follows:

\[
\text{MED}=2.400 \text{ TO } 2.500
\]

(Press R/S)

Two data values are displayed because when the number of data points is even, the median is not unique, but rather spans an interval from the one point listed above to the other. Many users may wish to simply take the middle of this interval as the median, but any point is technically correct in the interval. When the number of data points is odd, the median is unique and only one value will be displayed by the calculator.

10. After the median is displayed as described in step 9, the calculator will display the following statistics labeled as shown:

<table>
<thead>
<tr>
<th>Display</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN=2.100</td>
<td>Minimum Value</td>
</tr>
<tr>
<td>MAX=2.900</td>
<td>Maximum value</td>
</tr>
<tr>
<td>RNG=0.800</td>
<td>Range</td>
</tr>
</tbody>
</table>

11. At this point the calculator will respond with "CELL?" asking for the number of cells the user desires in the
histograms. If the number of cells is not significant at this point, the calculator will pick an appropriate number if the user simply presses run/stop. For this example, press:

R/S

12. Next the calculator responds with "WIDTH" asking for the width of the cells. Simply press run/stop if you do not wish to establish the width manually. Again, you may see the width the calculator will use by pressing the clear arrow key (Unless the width is an integer, you will also need to press FIX 3 to display the decimal properly if you wish to examine the width.) For this example, press:

R/S

13. The calculator will now display the cell frequency counts as an integer count followed by the next cell boundary. The leftmost cell boundary is set equal to the minimum value and is not explicitly output. If a data point falls exactly on a cell boundary, it is counted in the left cell.
For this example, the display will show:

<table>
<thead>
<tr>
<th>Display</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNT=2</td>
<td>Two observations between 2.1 (the minimum) and 2.26 (the cell boundary)</td>
</tr>
<tr>
<td>xx=2.260</td>
<td></td>
</tr>
<tr>
<td>CNT=3</td>
<td>Three observations between 2.26 (see above) and 2.42 (the next boundary)</td>
</tr>
<tr>
<td>xx=2.420</td>
<td></td>
</tr>
<tr>
<td>CNT=1</td>
<td>One observation between 2.42 (see above) and 2.58 (the next boundary)</td>
</tr>
<tr>
<td>xx=2.580</td>
<td></td>
</tr>
<tr>
<td>CNT=3</td>
<td>Three observations between 2.58 (see above) and 2.74 (the next boundary)</td>
</tr>
<tr>
<td>xx=2.740</td>
<td></td>
</tr>
<tr>
<td>CNT=1</td>
<td>One observation between 2.74 (see above) and 2.90 (the maximum)</td>
</tr>
<tr>
<td>xx=2.900</td>
<td></td>
</tr>
</tbody>
</table>

14. After the last cell boundary is displayed, the calculator will display "CMD" asking the user for the next command. Frequently, the user will wish to modify the histogram by changing the number of cells or the cell width. To recalculate the histogram cell frequencies without re-entering the data press:

`XEQ ALPHA AGAIN ALPHA`

If no further work with the histogram is desired, the user may view the order statistics simply by pressing run/stop.
## USER INSTRUCTIONS: SINGLE VARIABLE STATISTICS

<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SET SIZE (Nnn=16+NUMBER OF DATA POINTS)</td>
<td>XEQ &quot;SIZE</td>
<td>&quot;Nnn</td>
<td>UP TO nnn = 235</td>
</tr>
<tr>
<td>2</td>
<td>CALL THE PROGRAM (&quot;STAT1 IS FOR REGULAR USE) (&quot;S1 IS FOR CLASSROOM USE)</td>
<td>XEQ &quot;STAT1 - or - &quot;S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ENTER THE NUMBER DATA POINTS.</td>
<td>N?</td>
<td>input</td>
<td>R/S</td>
</tr>
<tr>
<td>4</td>
<td>ENTER THE DATA X1?, X2? etc.</td>
<td>input</td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For mistakes or to review the data see last two steps below.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHEN VERIFY MODE IS SET ON (SET BY FLAG 05 ON) AFTER EACH DATA POINT IS ENTERED, THE VALUE WILL BE ECHOED BACK BY THE CALCULATOR.</td>
<td>x1=xx etc.</td>
<td>R/S - or - correct value</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SUMMARY STATISTICS ARE CALCULATED WHEN ALL DATA HAS BEEN ENTERED.</td>
<td>XBAR=xx</td>
<td>R/S</td>
<td>mean</td>
</tr>
<tr>
<td></td>
<td>SSM = xx</td>
<td>R/S</td>
<td>sum of sq dev from mean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STANDARD DEVIATION SX=xx</td>
<td>R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SKEWNESS SKEW=xx</td>
<td>R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KURTOSIS KURT=xx</td>
<td>R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CALCULATOR WILL AUTOMATICALLY SORT DATA POINTS. AND THEN DISPLAY:</td>
<td>PRGM</td>
<td>STANDBY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEDIAN (note if N is even the median is not unique and an interval is displayed)</td>
<td>MED=xx</td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MINIMUM</td>
<td>MIN=xx</td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAXIMUM</td>
<td>MAX=xx</td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>RNG=xx</td>
<td>R/S</td>
<td></td>
</tr>
</tbody>
</table>
### User Instructions: Single Variable Statistics

<table>
<thead>
<tr>
<th>Step</th>
<th>Explanation</th>
<th>See</th>
<th>Press</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>User option to enter number of histogram cells. No input is required.</td>
<td>CELL?</td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R/S =-</td>
<td>INPUT N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>User option to enter width of histogram cells. (Has precedence over number of cells if a width is entered.)</td>
<td>WIDTH?</td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R/S =-</td>
<td>INPUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Calculate histogram (output data about each cell from left to right.)</td>
<td>CNT=II</td>
<td>R/S</td>
<td>CELL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FREQ</td>
<td>COUNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XX=XX</td>
<td>R/S</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X-VALUE</td>
<td>LIMIT</td>
</tr>
<tr>
<td>10</td>
<td>Accept next command</td>
<td>CMD</td>
<td>ENTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NEXT CMD</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Recalculate histogram</td>
<td></td>
<td>XEQ</td>
<td>&quot;AGAIN&quot;</td>
</tr>
<tr>
<td>12</td>
<td>Edit an input value at any time prior to data sort.</td>
<td>XEQ</td>
<td>&quot;HSC&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;SC&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>After input of new value calculator will return to data input or calculation of summary stats as appropriate.</td>
<td></td>
<td>INPUT</td>
<td>WILL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>POINT</td>
<td>REMOVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NUMBER</td>
<td>POINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X?</td>
<td>INPUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CORRECT</td>
<td>VALUE</td>
</tr>
</tbody>
</table>
USER INSTRUCTIONS: SINGLE VARIABLE STATISTICS

<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>REVIEW DATA POINTS (OR REVIEW ORDER STATS AFTER SORT.)</td>
<td></td>
<td>XEQ #SR</td>
<td></td>
</tr>
</tbody>
</table>
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
{
  STAT1
}

1 LBL "STAT1" {RECOMMENDED ENTRY POINT}
2 CF 05 {SET VERIFY MODE OFF}
3 SP 26 {ENABLE AUDIO}
4 GTO "SS"

{
  S1
}

5 LBL "S1" {ENTRY POINT FOR CLASSROOM USE}
6 SP 05 {SET VERIFY MODE ON}
7 CF 26 {DISABLE AUDIO TONES}
8 CF 21 {SET TO STOP DURING VERIFICATION}

{
  "SS"
}

9 LBL "SS" {ENTRY POINT FOR USER SET OPTIONS}
10 CF 29 {NO DIGIT GROUPING}
11 &REG 10 {ESTABLISH STATISTICAL REGISTERS}
12 CF 06 {USED BY DATA REVIEW FUNCTION}
13 CF 08 {USED BY DATA EDITING FUNCTION}

00 DATA ENTRY (F06-CLEAR)
   AND
   DATA REVIEW (F06-SET)

14 LBL 00
15 STO 04 {ESTABLISH INDIRECT ADDRESS BASE REG.}
16 STO 00 {INITIALIZE DATA ENTRY POINTER}
17 RCL 15 {NUMBER DATA POINTS (LAST PROBLEM)}
18 CLE "Y?" {CLEAR MEANS DATA ENTRY, NOT REVIEW}
19 FC? 06 {PROMPT}
20 / {SET UP LOOP COUNTER FOR DATA POINTS
21 +
22 STO 01

58
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

DATA ENTRY LOOP

01

(INCREMENT DATA STORAGE POINTER)

(RECALL DATA VALUE)

(TEMP STORAGE FOR LABEL)

(IS THIS REVIEW OF DATA PREV. ENTERED?)

(PROMPT USER FOR NEXT DATA VALUE)

(STORE THE DATA VALUE)

(NO VERIFICATION OF DATA DESIRED?)

(FOLLOWING IS THE VERIFICATION ROUTINE)

(RECALL THE LABEL)

(RECALL THE STORED DATA)

(CLEAR DATA ENTRY FLAG)

(WILL STOP FOR DATA ENTRY IF F21 SET)

(WAS THERE NO DATA CHANGE DURING VIEW?)

(IF THERE WAS A NEW VALUE, THEN RECORD)

(IT AND GO BACK AND RE-VERIFY THE DATA.)

(FOLLOWING IS THE STATISTICAL ACCUM.)

(STORES SIGMA X)

(STORES SIGMA X-SQUARED)

(STORES SIGMA X-CUBED)

(STORES SIGMA X-FOURTH-POWER)

(IS THIS A DATA REVIEW?)

(IF DATA ENTRY, INCREMENT INPUT CNTR.)

(AT END OF DATA ENTRY, RECALL INPUT)

(COUNTER, WHICH IS A NUMBER EQUAL ONE)

(MORE THAN NUMBR POINTS)
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
{ "SM" }  CALCULATION OF SUMMARY STATS

71 LBL "SM"  (ENTRY ASSUMES X-REGISTER HAS A NUMBER
72 INT        {1 MORE THAN NUMBER OF DATA POINTS.
73 1
74 STO 15    (STORES THE NUMBER OF DATA POINTS
75 "MEAN
76 XEQ 97    (CALL AN OUTPUT LABELING ROUTINE
77 STO 03    (TEMP STORE FOR XBAR
78 RCL 11    (RECALL SIGMA X-SQUARED
79 RCL 03    (RECALL XBAR
80 X^2       (RECALL NUMBR POINTS
81 *         (TEMP STORE FOR SUM OF SQUARED
82 RCL 09    (DEVIATIONS ABOUT THE MEAN
83 RCL 15    (NUMBER POINTS
84 /          (CAN NOT USE SDEV FUNCTION BECAUSE OF
85 'QRT       (NON-STANDARD USE OF REGISTERS 12-14
86 "SX
87 XEQ 97    (STANDARD DEVIATION
88 RCL 09    (SUM OF SQ DEVIATION ABOUT MEAN
89 RCL 15    (NUMBER POINTS
90 /          (SECOND MOMENT
91 XEQ 97    (SIGMA X-CUBED
92 RCL 12    (SIGMA X-SQUARED
93 RCL 11    (XBAR
94 *          (NUMBER POINTS
95 RCL 15    (XBAR
96 RCL 03    (XBAR
97 /          (THIRD MOMENT
98 XEQ 97    (SECOND MOMENT
99 RCL 05    (SIGMA X-FOURTH-POWER
100 RCL 13    (SIGMA X-CUBED
101 RCL 12    (XBAR
102 *          (OUTPUT THE SKEWNESS OF THE DATA
103 *          (SIGMA X-FOURTH-POWER
104 +          (SIGMA X-CUBED
105 STO 06     (XBAR
106 RCL 05    (SECOND MOMENT
107 RCL 13    (SIGMA X-FOURTH-POWER
108 RCL 03    (SIGMA X-CUBED
109 +          (XBAR
110 /          (OUTPUT THE SKEWNESS OF THE DATA
111 X<>Y
112 "SKEW    (SIGMA X-FOURTH-POWER
113 XEQ 97    (SIGMA X-CUBED
114 RCL 13    (XBAR
115 RCL 05    (SECOND MOMENT
116 RCL 13    (SIGMA X-FOURTH-POWER
117 RCL 03    (SIGMA X-CUBED
118 +          (XBAR
119 "SKEW
120 XEQ 97    (SIGMA X-FOURTH-POWER
121 RCL 13    (SIGMA X-CUBED
122 RCL 03    (XBAR
123 RCL 05    (SECOND MOMENT
124 RCL 13    (SIGMA X-FOURTH-POWER
125 RCL 03    (SIGMA X-CUBED
126 +          (XBAR
127 "SKEW
128 XEQ 97    (SIGMA X-FOURTH-POWER
129 RCL 13    (SIGMA X-CUBED
130 RCL 03    (XBAR
131 RCL 05    (SECOND MOMENT
132 RCL 13    (SIGMA X-FOURTH-POWER
133 RCL 03    (SIGMA X-CUBED
134 +          (XBAR
135 "SKEW
136 XEQ 97    (SIGMA X-FOURTH-POWER
137 RCL 13    (SIGMA X-CUBED
138 RCL 03    (XBAR
139 RCL 05    (SECOND MOMENT
140 RCL 13    (SIGMA X-FOURTH-POWER
141 RCL 03    (SIGMA X-CUBED
142 +          (XBAR
143 "SKEW
144 XEQ 97    (SIGMA X-FOURTH-POWER
145 RCL 13    (SIGMA X-CUBED
146 RCL 03    (XBAR
147 RCL 05    (SECOND MOMENT
148 RCL 13    (SIGMA X-FOURTH-POWER
149 RCL 03    (SIGMA X-CUBED
150 +          (XBAR
151 "SKEW
152 XEQ 97    (SIGMA X-FOURTH-POWER
153 RCL 13    (SIGMA X-CUBED
154 RCL 03    (XBAR
155 RCL 05    (SECOND MOMENT
156 RCL 13    (SIGMA X-FOURTH-POWER
157 RCL 03    (SIGMA X-CUBED
158 +          (XBAR
159 "SKEW
160 XEQ 97    (SIGMA X-FOURTH-POWER
161 RCL 13    (SIGMA X-CUBED
162 RCL 03    (XBAR
163 RCL 05    (SECOND MOMENT
164 RCL 13    (SIGMA X-FOURTH-POWER
165 RCL 03    (SIGMA X-CUBED
166 +          (XBAR
167 "SKEW
168 XEQ 97    (SIGMA X-FOURTH-POWER
169 RCL 13    (SIGMA X-CUBED
170 RCL 03    (XBAR
171 RCL 05    (SECOND MOMENT
172 RCL 13    (SIGMA X-FOURTH-POWER
173 RCL 03    (SIGMA X-CUBED
174 +          (XBAR
175 "SKEW
176 XEQ 97    (SIGMA X-FOURTH-POWER
177 RCL 13    (SIGMA X-CUBED
178 RCL 03    (XBAR
179 RCL 05    (SECOND MOMENT
180 RCL 13    (SIGMA X-FOURTH-POWER
181 RCL 03    (SIGMA X-CUBED
182 +          (XBAR
183 "SKEW
184 XEQ 97    (SIGMA X-FOURTH-POWER
185 RCL 13    (SIGMA X-CUBED
186 RCL 03    (XBAR
187 RCL 05    (SECOND MOMENT
188 RCL 13    (SIGMA X-FOURTH-POWER
189 RCL 03    (SIGMA X-CUBED
190 +          (XBAR
191 "SKEW
192 XEQ 97    (SIGMA X-FOURTH-POWER
193 RCL 13    (SIGMA X-CUBED
194 RCL 03    (XBAR
195 RCL 05    (SECOND MOMENT
196 RCL 13    (SIGMA X-FOURTH-POWER
197 RCL 03    (SIGMA X-CUBED
198 +          (XBAR
199 "SKEW
200 XEQ 97    (SIGMA X-FOURTH-POWER
201 RCL 13    (SIGMA X-CUBED
202 RCL 03    (XBAR
203 RCL 05    (SECOND MOMENT
204 RCL 13    (SIGMA X-FOURTH-POWER
205 RCL 03    (SIGMA X-CUBED
206 +          (XBAR
```

60
HP41C SOURCE CODE:  SINGLE VARIABLE STATISTICS

* * *
125 RCL 03
126 X<>Y
127 RCL 11
128 +
129 RCL 15
130 /
131 RCL 03
132 +
133 X
134 STO 07
135 RCL 05
136 X<>Y
137 KURT=
138 XEQ 97
139 RST 09
140 XEQ 98
141 RC 00
142 RCL 15
143 2
144 PRC
145 X=0?
146 SF 00
147 LASTX
148 .5
149 +
150 RCL 04
151 +
152 MED=
153 ARCL MOV X
154 FC? 00
155 GT0 05
156 "=" TO
157 PROMPT
158 1
159 +
160 "="
161 ARCL MOV X
162 LBL 05
163 PROMPT

(xbar)

(sigma x-squared)

(number points)

(xbar)

(fourth moment)

(second moment)

(output the kurtosis)

(short form would not compute stats)

(which require sorted data)

(call a data sorting routine)

(initialize temp flag used to check)

(even or odd number of data points)

(was it an even number of points?)

(if was even number, set flag)

(computing address of median)

(address base register)

(x-reg now has address of median)

(even number points implies the median)

(not unique, but spans an interval)

(display the left boundary of median)

(x-reg points to right bound of median)
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

({AGAIN

DISPLAY HISTOGRAM

} AGAIN

175 LBL "AGAIN

176 RCL 04

177 +

178 RCL IND X

179 "MIN

180 XEQ 97

181 STO 09

182 RCL 04

183 RCL 15

184 +

185 RCL IND X

186 "MAX

187 XEQ 97

188 RCL Z

189 RCL 2

190 -

191 STO 08

192 "RNG

193 XEQ 97

194 RCL 00

195 RCL 15

196 RCL 04

197 +

198 IE3

199 RCL 04

200 RCL 04

201 +

202 +

203 RCL 01

204 STO 01

205 RCL 15

206 LN

207 +

208 IE3

209 FIX 0

210 RND

211 "CELL?

212 PROMPT

213 RCL 08

214 5<>

215 "%WIDTH

216 PROMPT

217 STO 08

218 BCL 08

219 "CELL

220 BCL 08

221 STO 09

222 STO 02

223 LBL 02

224 "NEXT DATA POINT

225 BCL IND 01

226 BCL 09

227 FIX 3

228 RND

ADDRESS BASE REGISTER

RECALL THE FIRST ORDER STAT

CALL AN OUTPUT LABELING ROUTINE

HOLDS STARTING (LEFTMOST) X BOUNDARY

ADDRESS BASE REGISTER

NUMBER OF DATA POINTS

RECALL THE N-TH ORDER STATISTIC

DISPLAY THE MAX VALUE OBSERVED

MIN

TEMP STORE FOR THE RANGE

DISPLAY THE RANGE

INITIALIZE TEMP FLAG TO MARK LAST BAR

ADDRESS BASE REGISTER

NUMBER DATA POINTS

COMPUTING INDEX LOOP COUNTER

ADDRESS BASE REGISTER

801 SET TO ADDRESS AND LOOP THRU DATA

NUMBER POINTS

DEFAULT NUMBER OF BARS IS 2*LN(N)

VALUE IS ROUNDED NOT TRUNCATED

USER HAS OPTION TO CHANGE NUMBER CELLS

RANGE

USER HAS OPTION TO CHANGE CELL WIDTH

NOW HOLDS CELL WIDTH NOT RANGE

CELL WIDTH

UPPER LIMIT OF CURRENT CELL COUNTED

INITIALIZE CELL COUNTER

NEXT DATA POINT

CELL UPPER LIMIT
**HP41C Source Code:**

229 X<Y?
230 GTO 08
231 1
232 ST+ 02
233 ISG 01
234 GTO 07
235 SF 00
236LBL 08
237 "CNT
238 RCL 02
239 FIX 0
240 XEQ 97
241 FIX 3
242 "XX
243 RCL 09
244 XEQ 97
245 FC?C 00
246 GTO 06
247 "CHD
248 AVIEW
249 RTN

**SINGLE VARIABLE STATISTICS**

- (DATA POINT LESS THAN UPPER LIMIT)
- (INCREMENT THE CELL COUNTER)
- (PREPARE TO LOOK AT NEXT DATA POINT)
- (SET FLAG FOR OUTPUT OF LAST BAR)
- (OUTPUT THE CELL FREQUENCY COUNT)
- (OUTPUT CELL BOUNDARY—LOWER LIMIT)
- (IS THIS THE LAST BAR?)
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
{ "SR
    REVIEW THE DATA

250 LBL "SR
251 SF 06 (SETS MODE FOR REVIEW NOT QUERY
252 GTO 00

{ "SC
    EDIT THE DATA

253 LBL "SC
254 SF 08 (SET TO EDIT MODE
255 CP 06 (SET TO QUERY FOR CORRECT VALUE
256 RCL 00 (CURRENT INPUT ADDRESS POINTER
257 STO 05 (SAVE TO ENABLE RETURN TO DATA ENTRY
258 RCL 01 (CURRENT INPUT INDEX LOOP COUNTER
259 STO 06 (SAVE TO ENABLE RETURN TO DATA ENTRY
260 "POINT?
261 PROMPT
262 STO 01
263 RCL 04 (ADDRESS BASE REGISTER
264 * (COMPUTER ADDRESS OF DATA TO BE EDITED
265 STO 00
266 RCL IND 00 (RECALL THE OLD VALUE
267 ST- 10 (CORRECT SIGMA X
268 X^2
269 ST- 11 (CORRECT SIGMA X-SQUARED
270 LASTX
271 *
272 ST- 12 (CORRECT SIGMA X-CUBED
273 LASTX
274 X^3
275 ST- 13 (CORRECT SIGMA X-FOURTH-POWER
276 XEQ 02 (CALL DATA ENTRY AS A SUBROUTINE
277 CP 08 (FOLLOWING RESTORES DATA ENTRY
278 RCL 05 (INPUT ADDRESS REGISTER VALUE
279 STO 00
280 RCL 06 (INPUT LOOP COUNTER
281 STO 01
282 1
283 - (TEST TO SEE IF ALL DATA ALREADY INPUT
284 ISG X
285 GTO 02 (IF NOT, BRANCH TO THE INPUT LOOP
286 GTO "SM

64
```
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

\begin{verbatim}
97 OUTPUT LABELING ROUTINE

287 LBL 97
288 M = 
289 ARCL X
290 PROMPT
291 RTN
\end{verbatim}
HP41C SOURCE CODE:  SINGLE VARIABLE STATISTICS

98 SHELL SORT

292 LBL 98
293 RCL 15
294 STO 01
295 2
296 RCL 01
(RECALL NUMBER OF DATA POINTS
DEFINE A = "MIDPOINT"
(RECALL MIDPOINT

A = INT(A/2)
(TEST TO SEE IF LIST SORTED

(B = 1  -- RESET LEFT SHELL BOUNDARY

STACK TABLE FOLLOWS:

C=B  B
D=C*A  B
A  C
C
B

D
B

X(D)  ADDR D  B
C  X(D)  ADDR D  ;B
BASE  C  X(D)  ADDR D  ;ADDR D
ADDR C  X(D)  ADDR C  ADDR D
X(C)  X(D)  ADDR C  ADDR D

X<=Y?

FOLLOWING INTERCHANGES X(C) AND X(D)

X(D)  X(C)  ADDR C  ;ADDR D
X(C)  X(D)  ADDR C  ;ADDR D

X(C)  X(D)  ADDR C  ADDR D

C

C=C-A

N
N
E

B=N-A
B
B+1
B=B+1
**HP41C Source Code:**

**Single Variable Statistics**

```
THIS PROGRAM USES THE FOLLOWING REGISTERS:

R00 -- INPUT DATA ADDRESS POINTER
R01 -- LOOP INDEX COUNTER
        (USED BY DATA ENTRY AND SORT ROUTINES)
R02 -- TEMP REGISTER
        (CELL FREQUENCY COUNT IN HISTO RTN)
        (AND SHELL BOUNDARY IN SORT ROUTINE)
R03 -- TEMP REGISTER
        (INPUT LABEL IN DATA INPUT ROUTINE)
        (XBAR IN SUMMARY STAT ROUTINE)
        (AND POINTER IN SORT ROUTINE)
R04 -- INDIRECT ADDRESS BASE
R05 -- SECOND MOMENT (POPULATION VARIANCE)
R06 -- THIRD MOMENT
R07 -- FOURTH MOMENT
R08 -- HISTOGRAM CELL WIDTH
R09 -- TEMP REGISTER
        (Holds SUM OF SQ DEVIATION ABOUT MEAN)
        (AND HISTOGRAM CELL UPPER LIMIT)
R10 -- SUM OF X VALUES
R11 -- SUM OF X-SQUARED VALUES
R12 -- SUM OF X-CUBED VALUES
R13 -- SUM OF X RAISED TO THE FOURTH POWER
R14 -- NOT USED BUT SET TO ZERO BY CLR
R15 -- NUMBER DATA POINTS (SET BY &+)
R16.... R228 RAW DATA POINTS
        -- IN NATURAL SEQUENCE BEFORE SORT
        -- AS ORDER STATISTICS AFTER SORT

THIS PROGRAM USES THE FOLLOWING FLAGS:

P00 -- TEMP FLAG (USED IN EDIT AND HISTO RTNS)
P05 -- VERIFY MODE (EVERY DATA POINT ECHOED)
P06 -- INDICATES REVIEW OF DATA NOT QUERY MODE
P08 -- INDICATES EDITING A DATA POINT
P21 -- PRINTER ENABLE (STOPS CALCULATOR
        DURING VIEW INSTRUCTION)
P26 -- AUDIO ENABLE
```

340 END
APPENDIX B

LINEAR PROGRAMMING EXAMPLE

INTRODUCTION:

Linear programming is an operations research technique normally associated with computerized data bases and the largest computers. Because of the complexity of the computer programs for linear programming and the large amount of data associated with most real world problems, calculators have not been widely used for this application. With the increased memory capacity of the HP41CV, however, it is now possible to offer a calculator program which can solve interesting small scale linear programs. Of value primarily as an educational aid, this program will also be able to solve many small scale problems found at Army division, brigade and battalion level. The text by Hillier and Lieberman [Ref. 8: pp. 16-66] is recommended as an introduction to the theory of linear programming as used by the program given in this appendix. Use of the program requires the user to formulate the linear programming problem; set up a Simplex tableau in standard form including adding slack, surplus and artificial variables as required; and interpret
the final tableau including the calculation of the values associated with the variables in the final basis. Using the tableau form of the Simplex algorithm, the calculator performs both phase I (to obtain a feasible solution) and phase II (to obtain an optimal solution) to solve the linear programming problem. The calculator automatically determines the pivot column and pivot row for each pivot step. Infeasible and unbounded problems are automatically identified for the user by the program. There is no explicit handling of variables with upper bounds.

PROGRAM DESCRIPTION:

The program is written as a series of subroutines, each of which performs a major step in the Simplex algorithm. To provide clarity to the user, alphabetic labels have been retained to identify the subroutines in lieu of faster and more memory efficient numeric labels. The alphabetic labels have not been retained for use as program entry points and may be changed to numeric labels at the option of the user. The program has two entry points, "LP" for running a new problems and "ALP" for reviewing data previously entered.

Subroutine "FINDQ" determines the pivot column by selecting the variable in the objective function with the most negative "price." If "FINDQ" discovers at least one
negative value in the objective function, then the tableau column number associated with the most negative value will be stored in register 05. Upon return from "FINDQ," the main routine tests register 05 to see if it contains a non-zero entry. If the entry is zero, it means that no further pivots will improve the value of the objective function, and the Simplex algorithm halts. If the program was in phase I (flag 11 clear) and the value of the phase I objective function is reduced to zero, then a feasible solution has been found and the program will automatically proceed to phase II to discover an optimal solution.

Subroutine "FINDP" determines which variable will leave the current basis by performing a minimum positive ratio test along the pivot column. In this way, the pivot row is determined. The row number of the pivot row is stored in register 06. Upon return from subroutine "FINDP," the main routine tests register 06 to see if it contains a non-zero entry. If the entry is zero, it means that the problem is unbounded and the Simplex algorithm halts. Such an unbounded condition is most likely caused by an error in the problem formulation.

Having determined the pivot column and the pivot row, subroutine "PIVOT" performs the actual Simplex pivot
operation. To speed calculation register 00 is used as a temporary register to hold the reciprocal of the pivot element. Note that the pivot row is handled separately from the other rows in the tableau. Flag 04 is used to provide the option of stopping calculation after every pivot. When this flag is set, the program will halt to allow the user to review the status of the tableau with the "ALP" function.

Subroutine "CHECK" has two primary functions. First, it is used to verify that the designated basic variables are in row elimination form prior to the start of the Simplex algorithm. This means that the basic variable must have a coefficient of 1 in the row in which it is basic and zero's in all other rows. The second function of check is to prepare the objective function for phase I, if the initial basis contains artificial variables as indicated by one or more minus signs in the "JB" vector.

Three subroutines are used to query the user for input data. Subroutine "READMN" queries the user for the number of constraints and decision variables in the problem and verifies the calculator memory is set to contain all the data necessary to solve the problem. Subroutine "READJB" queries the user for a column vector of pointers which indicate which variable is currently basic in each row. When
entering this vector of pointers, the user indicates artificial variables with a minus sign. Subroutine "READA" queries the user for the values in the initial Simplex tableau including the slack and surplus variables and the right hand side and objective function.

Several other service routines also are provided in this program. Memory size verification is done by subroutine "SIZE," which is called from within "READMN." Subroutine "IN" is used to query the user for data entry and is called by all of the data input routines. Subroutine "NXT" initializes registers which contain frequently used quantities such as the total size of the tableau for phase I and phase II. Subroutine "INIT" clears the calculator memory and sets flags and program constants appropriately for input of a new problem. Subroutine "SETL" establishes the loop counters used repeatedly within almost every other subroutine. Subroutine "ERR1" displays an appropriate error message if a data entry error is detected.

SAMPLE PROBLEM:

A division assistant G4 is planning an ammunition upload plan. There are four types of tank munitions to consider,
including:

\[ A = \text{Discarding Sabot Rounds} \]
\[ B = \text{High Explosive Anti-Tank Rounds} \]
\[ C = \text{Phosphorous Munitions} \]
\[ D = \text{Machine Gun Ammunition} \]

Based on the Commander's guidance the assistant G4 is to consider the sabot rounds as twice as important as the HEAT rounds, which in turn are themselves twice as important as a unit amount of phosphorous munitions and machine gun ammunition. His mission then, is to maximize:

\[ Z = 4A + 2B + C + D \]

He is, however, constrained by the following factors:

1. There can be no more than 30 units of both sabot and HEAT munitions combined.
2. There can be no more than 50 units of all types of ammunition combined.
3. There must be at least 30 units of HEAT and phosphorous munitions combined.
4. There must be at least 5 units of machine gun ammunition.

These constraints may be expressed as:

\[
\begin{align*}
A + B & \leq 30 \\
A + B + C + D & \leq 50 \\
B + C & \geq 30 \\
D & \geq 5
\end{align*}
\]

Based on the Commander's guidance and the constraints listed above, formulate an optimum load plan. Fractional units are permitted.
SOLUTION:

1. Before beginning with the calculator, the first step is to layout the tableau in standard form. This step and the last step of interpreting the final tableau require working knowledge of linear programming as explained in Hillier and Liberman [Ref. 8: pp. 16-66]. The standard form of the tableau is:

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

In this tableau, H1 and H2 are surplus variables; S1 and S2 are slack variables; and A1 and A2 are artificial variables.

2. The first step with the calculator is to set the size of the calculator's data memory. This program requires 20 registers for temporary storage, 1 register for each tableau element, and 1 register for each row to hold the pointer to the basic variable for that row. Thus, if M is the number of constraints and N is the number of variables including slack, surplus and artificial variables, then the total data...
storage requirement is:

\[
\text{storage required} = 21 + M + ((N + 1) \times (M + 2))
\]

As mentioned in the program description, the "SIZE" subroutine will automatically verify that the user has allocated enough data storage to solve the problem. The length of the program is such that 177 data storage registers is the maximum number of data storage registers that can be allocated. Thus, linear programs with 7 constraints and 15 decision variables can be solved with this program. For this example, press:

\[
\text{XEQ } \text{ALPHA } \text{SIZE } \text{ALPHA } 175
\]

3. Call for execution of the program with a new data set. Press:

\[
\text{XEQ } \text{ALPHA } \text{LP } \text{ALPHA}
\]

4. The calculator will respond with the prompt "NUM ROWS?" asking for the number of constraints in the linear program formulation. In this example, press:

\[
4 \ R/\S
\]

5. The calculator will respond with the prompt "NUM COLS?" asking for the number of variables in the problem. The user
must count the number of slack, surplus and artificial variables in this total. In this example, press:

10 R/S

6. The calculator will respond with the prompt "BASIC 1 ?" asking for the variable number of the variable which is basic in the first row. One of the major features of this program is that the basic variables need not be in the rightmost positions in the tableau. Thus, if a tableau were given in which some pivots had already been performed, the program could resume operation immediately. In this example, press:

7 R/S

In a similar fashion, the calculator will then query the user for the variable number of the variables which are basic in the remaining rows.

For this example:

<table>
<thead>
<tr>
<th>See</th>
<th>Respond With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic 2 ?</td>
<td>8 R/S</td>
</tr>
<tr>
<td>Basic 3 ?</td>
<td>9 CHS R/S</td>
</tr>
<tr>
<td>Basic 4 ?</td>
<td>10 CHS R/S</td>
</tr>
</tbody>
</table>

Notice that because the basic variables in rows three and four are artificial variables, the variable number is entered as a negative number. This signals the calculator
that these variables must be driven from the basis in order to reach an initial feasible solution.

7. Next, the calculator will respond with "T1,1?" asking for the first element in the tableau. The user should enter the numbers in the tableau using the digit entry keys and pressing run/stop after every entry. Notice that the right hand side and the objective function will be entered with the appropriate index in the tableau as shown in step 1 above. The user must insure that the objective function is in standard form with the appropriate sign for each coefficient—in this example each coefficient is negative.

8. After the last element in the tableau has been entered, the calculator will begin to automatically perform the Simplex algorithm. If the user wishes to stop the calculator after every pivot, he may at any time press:

\[ R/S \quad SF \quad 04 \quad R/S \]

If this flag is set, the calculator will stop and display the pivot number after every pivot is completed.

9. When the Simplex algorithm can no longer improve the objective function, the calculator will stop and display the value of the objective function. In this example, the
calculator will stop after approximately three minutes and display:

\text{VALUE}=110.000

10. At this point, the user must use entry point "ALP" to determine the status of the final tableau. For this example, press:

\text{XEQ ALPHA ALP ALPHA}

Then by sequentially pressing the run/stop and clear arrow keys, the basic variables and tableau entries will be displayed. For example, in this problem:

<table>
<thead>
<tr>
<th>See</th>
<th>Press</th>
<th>See</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC 1?</td>
<td>CLX</td>
<td>2</td>
<td>Variable 2 is basic in the first row.</td>
</tr>
<tr>
<td>BASIC 2?</td>
<td>CLX</td>
<td>1</td>
<td>Variable 1 is basic in the second row.</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then for the elements of the tableau:

<table>
<thead>
<tr>
<th>See</th>
<th>Press</th>
<th>See</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1,1?</td>
<td>CLX</td>
<td>0.000</td>
<td>Tableau entry</td>
</tr>
<tr>
<td>T1,2?</td>
<td>CLX</td>
<td>1.000</td>
<td>Tableau entry</td>
</tr>
</tbody>
</table>

11. After the calculator is finished, it remains for the user to interpret the final tableau. Again, the reference by Hillier and Lieberman [Ref. 8: pp. 16-66] is of primary value. In particular, the user must be able to determine the value of the final decision variables based upon what
variables are in the basis, and what the final "right hand side" values are for each row. For this example, the final tableau is:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RHS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>

Thus, the solution may be interpreted as 15 units each for munitions A, B, and C and 5 units for munition D.
<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SET SIZE</td>
<td></td>
<td>XEQ</td>
<td>UP TO</td>
</tr>
<tr>
<td></td>
<td>(N,M) =</td>
<td></td>
<td>&quot;SIZE</td>
<td>N,N</td>
</tr>
<tr>
<td></td>
<td>21+M*(N+1)(M + 2)</td>
<td></td>
<td>NNN</td>
<td>= 177</td>
</tr>
<tr>
<td></td>
<td>WHERE M=NUM RNWS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AND N=NUM COLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CALL THE PROGRAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ENTER THE NUMBER</td>
<td>NUM ROW</td>
<td>INPUT N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OF CONSTRAINTS</td>
<td></td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ENTER THE NUMBER</td>
<td>NUM COL</td>
<td>INPUT N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OF VARIABLES</td>
<td></td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCLUDE SLACKS,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SURPLUS &amp; ARTIF.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ENTER CURRENT</td>
<td>BASIC 1</td>
<td>INPUT VAR #</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BASIC VARIABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NUMBERS BY ROW</td>
<td></td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ENTER TABLEAU</td>
<td>TI,1?</td>
<td>INPUT R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUES FOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MISTAKES OR</td>
<td>ETC.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TO REVIEW THE DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEE LAST STEP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BELOW.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TO FORCE THE CALC-</td>
<td>R/S</td>
<td>SF 04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ULATOR TO STOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AFTER EACH PIVOT</td>
<td></td>
<td>R/S</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SIMPLEX COMPLETED:</td>
<td>VALUE</td>
<td>FINAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPTIMAL SOLUTION</td>
<td>XX.XX</td>
<td>OBJ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FOUND</td>
<td></td>
<td>FUNCTION</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>VALUE</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SIMPLEX COMPLETED:</td>
<td>INFEAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROBLEM IS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INFEASIBLE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**USER INSTRUCTIONS: LINEAR PROGRAMMING**

<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td><strong>SIMPLEX COMPLETED:</strong> PROBLEM IS UNBOUNDED.</td>
<td><strong>UNBOUND</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>TO REVIEW VALUES IN TABLEAU AT ANY TIME INCLUDING FINAL TABLEAU.</td>
<td></td>
<td>**KEY **</td>
<td><strong>&quot;ALP&quot;</strong></td>
</tr>
<tr>
<td></td>
<td>AS PROMPTS APPEAR, DATA CAN BE CHANGED BY ENTERING NEW VALUE.</td>
<td><strong>BASIC 1</strong></td>
<td><strong>CLX</strong></td>
<td><strong>PROMPT WILL VANISH LEAVING DATA</strong></td>
</tr>
<tr>
<td></td>
<td><strong>WHAT IS CURRENTLY BASIC?</strong></td>
<td><strong>ETC.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WHAT ARE VALUES IN TABLEAU?</strong></td>
<td><strong>ETC.</strong></td>
<td><strong>CLX</strong></td>
<td><strong>PROMPT WILL VANISH LEAVING DATA</strong></td>
</tr>
<tr>
<td></td>
<td>OBTAIN VALUES OF FINAL SOLUTION FROM KNOWING WHICH VARS ARE BASIC AND VALUE OF RIGHT-HAND-SIDE FROM THE TABLEAU.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HP41C SOURCE CODE:  LINEAR PROGRAMMING

```
LBL "LP"

1 LBL "TP"  
2 LBL "ALP"
3 XEQ "READMN"
4 XEQ "READJB"
5 XEQ "READA"
6 XEQ "CHECK"
7 LBL 15
8 XEQ "FINDQ"
9 RCL 05
10 XEQ 35
11 FS? 11
12 GTO 25
13 RCL 04
14 RCL 10
15 RCL 12
16 RCL 07
17 RCL IND Y
18 ABS
19 X<Y?
20 "IMPEAS"
21 AVIEW
22 STOP
23 LBL 20
24 SF 11
25 RCL 04
26 RCL 08
27 RCL 12
28 *
29 STO 14
30 GTO 04
31 RCL 09
32 RCL 12
33 * 
34 +
35 STO 14
36 GTO 12
37 LBL 12
38 RCL 04
39 RCL 09
40 RCL 12
41 * 
42 RCL IND X
43 STO 00
44 "VALUE=
45 RCL 00
46 AVIEW
47 STOP
48 LBL 35
49 XEQ "FINDP"
50 XEQ 06
51 FS? 06
52 GTO 40
53 "UNBOUND"
```
HP41C SOURCE CODE:  LINEAR PROGRAMMING

55 ARCL 05
56 STOP
57LBL 40
58XEQ "PIVOT
59GTO 15
HP41C SOURCE CODE: LINEAR PROGRAMMING

(FINDQ)

60 LBL "FINDQ"
61 SF 01
62 STO 03
63 STO 05
65 RCL 11
66 RCL 01
67 XEQ *SETL
68 LBL 31
69 RCL 34
70 RCL 01
71 +
72 STO 00
73 RCL IND X
74 RCL 03
75 -
76 RCL 07
77 CHS
78 X<Y?
79 GTO 38
80 FC? 11
81 GTO 37
82 RCL 01
83 +
84 RCL IND X
85 ABS
86 RCL 07
87 X<Y?
88 GTO 38
89 LBL 37
91 RCL IND 00
92 STO 03
93 RCL 01
94 INT
95 STO 05
96 LBL 38
97 ISG 01
98 GTO 31
99 CF 01
100 RIN
HP41C SOURCE CODE:  LINEAR Programming

{ FINDP }  
{ FINDP }

101 LBL "FINDP"
102 SF 02
103 0
104 STO 06
105 1E20
106 STO 03
107 1
108 RCL 08
109 XEQ "SETL"
110 LBL 41
111 RCL 01
112 -
113 RCL 12
114 RCL 04
115 +
116 STO 00
117 RCL 05
118 RCL IND X
119 +
120 RCL IND X
121 STO 02
122 RCL 07
123 X<Y?
124 GTO 48
125 RCL 00
126 RCL 12
127 +
128 RCL IND X
129 RCL 02
130 RCL 00
131 /  
132 STO 00
133 RCL 03
134 -
135 RCL 07
136 RCL CHS
137 X<Y?
138 GTO 48
139 LBL 47
140 RCL 00
141 ISG 01
142 RCL 01
143 LBL 00
144 STO 06
145 LBL 48
146 ISE 01
147 GTO 41
148 CF 02
149 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

{ PIVOT }

LBL "PIVOT"
SF 03
RCL 06
1
RCL 12
* RCL 04
+ STO 03
RCL 05 +
RCL IND X
1/4 STO 00
RCL 12 XEQ "SETL"
LBL 51
RCL 03
RCL 01 +
RCL 00
STO IND Y
ISG 01
STO 51
1 RCL 09
PS? 11
PC? 11
1 +
1 XEQ "SETL"
LBL 52
RCL 06
RCL 01
INT
XEQ "SETL"
RCL 01
RCL 12
*
RCL 04
+ STO 16
RCL 05 +
RCL IND X
STO 00
RCL 12
XEQ "SETL"
HP41C SOURCE CODE:  LINEAR PROGRAMMING

204 LBL 53
205 CF 07
206 RCL 05
207 RCL 02
208 INT
209 X=0?
210 SF 07
211 RCL 03
212 +
213 RCL IND X
214 RCL 00
215 +
216 RCL 16
217 RCL 02
218 +
219 X<Y
220 ST - IND Y
221 FC? 07
222 GTO 54
223 0
224 STO IND Z
226 TSG 02
227 GTO 53
228 LBL 59
229 TSG 01
230 GTO 52
231 BCL 13
232 RCL 06
233 +
234 RCL 05
235 STO IND Y
236 1
237 ST + 17
238 CF 03
239 TONE 7
240 FC? 04
241 "PIVOT"
242 ARC 17
243 PRCH PT
245 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

{ READMN }

246 LBL "READMN"
247 STO 00
248 "NUM ROWS XEQ "IN
249 "NUM COLS XEQ "NXT
250 "NUM COLS XEQ "NXT
251 RCL 10
252 RCL 04
253 +
254 STO 00
255 STO IND 00
256 RCL 09
257 +
258 "SIZE? XEQ C 25
259 PROMPT
260 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

```
270 LBL "SIZE?"
271 "SIZE>="
272 ARCL X
273 -
274 SF 25
275 RCL IND X
276 RTN

278 LBL "NXT"
279 1
280 ST+ 00
281 ST+ IND 00
282 RTN

284 LBL "SETL"
285 1E03
286 /
287 1
288 ST+ IND Y
289 RTN

---
```
HP41C SOURCE CODE:  LINEAR PROGRAMMING

```
291 LBL "READJB
292 RCL 08
293 XEQ "SETL
294 RCL 13
295 STO 00
296 FIX 0
297 LBL 01
298 "BASIC",
299 ARCL 01
300 "",
301 ARCL 01
302 XEQ "IN
303 ISG 01
304 GTO 01
305 FIX 4
306 RTN
```
HP41C SOURCE CODE: LINEAR PROGRAMMING

{ READA }

307 LBL "REALA"
308 SF 10
309 1
310 RCL 09
311 XEQ "SETL"
312 LBL 11
313 1
314 RCL 12
315 XEQ "SETL"
316 RCL 12
317 FIX 0
318 "T"
319 ARCL 01
320 "-"
321 ARCL 02
322 FIX 4
323 LBL 13
324 RCL 01
325 1
326 RCL 12
327 *
328 RCL 02
329 +
330 RCL 04
331 +
332 -
333 STO 00
334 RCL 14
335 LBL 14
336 GTO 16
337 XEQ "IN"
338 GTO 17
339 LBL 16
340 "="
341 ARCL IND 00
342 AVIEW
343 IND 17
344 LBL 17
345 ISG 02
346 GTO 12
347 ISG 01
348 GTO 11
349 C? 10
350 RTN
352 RTN
CROSS COMPILER AND PROGRAMMING SUPPORT SYSTEM FOR THE HP41CV -- ETC

J N RICHMANN

SEP 81

UNCLASSIFIED
HP41C SOURCE CODE: LINEAR PROGRAMMING

(IN)

353 LBL "IN
354 CF 22
355 1
356 ST+ 00
357 RCL IND 00
358 ? PROMPT
359 STO IND 00
360 RTN

(INIT)

362 LBL "INIT
363 CLRG
364 CF 29
365 FTO 04
366 STO 07
367 19
368 STO 04
369 RTN

(ERR)

370 LBL "ERR
371 0
372 LN
373 EQ "READJB
HP41C SOURCE CODE: LINEAR PROGRAMMING

{ CHECK }

01 LBL "CHECK"
02 SF 11
03 RCL 12
04 XEQ "SETL"
05 RCL 09
06 RCL 12
07 *+:
08 STO 14
09 STO 15
10LBL 91
11RCL 14
12CL 01
130
14STO IND Y
15ISG 01
16GTO 91
171
18RCL 08
19XEQ "SETL"
20LBL 92
21CF 07
22RCL 13
23RCL 01
24+:
25RCL IND X
26X<0?
27SF 07
28ABS
29STO 00
30X=0?
31GTO "ERR1"
32X<Y?
33GTO "ERR1"
34X=0:
35RCL 09
36XEQ "SETL"
37LBL 93
38RCL 08
39GTO "ERR1"
40RCL 01
41INT
42X<0?
43SF 08
441
45RCL 12
46ACL 00

97
HP41C SOURCE CODE: LINEAR PROGRAMMING

428 +
429 RCL 04
430 + FC? 08
431 0
432 FS? 08
433 1
434 STO IND Y
435 ISG 02
436 GTO 93
437 FC? 07
438 GTO 98
439 CP 11
440 BCL 14
441 RCL 00
442 +
443 STO IND Y
444 BCL 12
445 XEQ ^SETL
446 LBL 96
447 RCL 01
448 INT 1
449 RCL 12
450 RCL 02
451 * RCL 04
452 + RCL IND X
453 *<Y
454 RCL IND Y
455 +
456 STO IND Y
457 ISG 02
458 GTO 96
459 LBL 98
460 ISG 01
461 GTO 92
462 PC? 11
463 RTRN
464 RCL 04
465 RCL 08
466 RCL 12
467 + 0
468 STO 14
469 MN
THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG ASSIGNMENTS MADE BY THIS PROGRAM:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R00</td>
<td>Temporary register; holds reciprocal of pivot element in subroutine pivot.</td>
</tr>
<tr>
<td>R01</td>
<td>Loop index counter</td>
</tr>
<tr>
<td>R02</td>
<td>Loop index counter</td>
</tr>
<tr>
<td>R03</td>
<td>Temporary register; holds min value in FINDQ; max value in FINDP; and is an intermediate address calculation value in pivot.</td>
</tr>
<tr>
<td>R04</td>
<td>Base register for indirect addressing (SET = 19)</td>
</tr>
<tr>
<td>R05</td>
<td>The pivot column number</td>
</tr>
<tr>
<td>R06</td>
<td>The pivot row number</td>
</tr>
<tr>
<td>R07</td>
<td>Effective zero level</td>
</tr>
<tr>
<td>R08</td>
<td>M = number of rows</td>
</tr>
<tr>
<td>R09</td>
<td>M plus 1</td>
</tr>
<tr>
<td>R10</td>
<td>M plus 2</td>
</tr>
<tr>
<td>R11</td>
<td>N = number of variables</td>
</tr>
<tr>
<td>R12</td>
<td>N plus 1</td>
</tr>
<tr>
<td>R13</td>
<td>Base register for the location of the vector JB which contains pointers to which variable is basic in each row.</td>
</tr>
<tr>
<td>R14</td>
<td>Row number of the objective function; set to M plus 1 or M plus 2 as determined by need for phase I.</td>
</tr>
<tr>
<td>R15</td>
<td>Base register for the location of the phase I objective function; if needed.</td>
</tr>
<tr>
<td>R16</td>
<td>Temporary register</td>
</tr>
<tr>
<td>R17</td>
<td>Number of pivots performed</td>
</tr>
<tr>
<td>R18</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>R19-R177</td>
<td>Data storage registers for elements of the tableau and the JB vector.</td>
</tr>
</tbody>
</table>

FLAGS

F01 - F03 = Subroutine execution flags. Because these flags are visible in the display they can be set when entering a major subroutine and cleared when leaving -- giving the user a visual indication of what is happening inside the calculator.

F01 -- Subroutine FINDQ
F02 -- Subroutine FINDP
F03 -- Subroutine PIVOT

F04 = When set, stops calculator after each pivot.
F07 = Used as temporary flag in pivot and check routines.
F10 = Used as a temporary flag in read routines.
F11 = When set, indicates phase II is in progress.
F29 = Controls format of display separator.
"LP" LINEAR PROGRAMMING
"LP" LINEAR PROGRAMMING
"LP" LINEAR PROGRAMMING
APPENDIX C

SUBROUTINES FOR READ ONLY MEMORY

INTRODUCTION:

The calculator subroutines described in this appendix perform functions which are frequently required by application programs and are therefore ideal candidates for use in a read only memory (ROM.) These routines were written especially to illustrate the differences between read only memory routines and routines designed for individual use via bar code or magnetic cards. These differences include more attention to entry and exit point options, an attempt to keep the size of the routines as compact as possible, and an attempt not to disturb the register stack if at all possible.

These common subroutines are provided separately from application programs because when more than one application program uses the routines, as is recommended for these programs, the use of a separate block of common functions saves space in the ROM overall. Also, by providing a convenient set of "macro" instructions, application programs can be constructed more quickly and easily. Because these
subroutines are used frequently, they have been individually optimized and tested to save memory space and execution time. By using these "macros" within an application program, the application programmer can be reasonably certain of their efficiency and reliability.

Almost all user/calculator interface is handled by these routines. There is one set of subroutines which assumes the user has a printer, and one set which does not. Printer instructions are preceded in the listings shown in the appendix by an (PRT: label. When not using these routines on read only memory, the user will load one set or the other (but not both), as appropriate for his/her application. In so doing, the user with the printer gets full benefit from it while the user without the printer pays no penalty in execution time or memory space for the calculator's print instructions. Also, to change from use of the printer to use of the calculator without it, the user needs only to read in the new common block—the application programs are retained in memory unchanged. The subroutines appear the same to any application program—giving the added benefit that any application program which uses them for input or output operations will automatically make good use of
the printer even if written by a programmer who did not explicitly consider a printer requirement.

When using these common subroutines, a discipline is enforced upon the application program concerning use of the calculator memory registers. This saves the programmer from having to plan his "register map" from scratch for each new program. Also, it insures compatibility across different application programs for similar data objects such as matrices and loop index counters. One of the most annoying problems with read only memory programs available from the calculator manufacturer is this lack of cross program compatibility. Conflict in the use of memory registers is the rule, rather than the exception; and it is frequently impossible to efficiently use more than one read only memory program as a subroutine within a user written program. A third reason why register assignment standards are advantageous is that they make it easier for the user of the calculator to remember the key register assignments and, if necessary, recall their contents during the execution of an application program.

Another function performed by this set of common subroutines is to simplify the use of indirect addressing—a critical goal on the HP41CV.
Because the common subroutines listed in this appendix are always called by application programs and never from the keyboard by the user, the typical user instructions are inappropriate. Instead, for the benefit of application programmers wishing to use the routines, the basic functions and structure of each are explained in subsequent sections of this appendix.

**Subroutine "IN"**

Subroutine "IN" is used as a general input and output interface between the user and the calculator. This subroutine has three alternative entry points which when called affect functions as follows:

- **IN**—Input mode (displays a question mark query)
- **IO**—Output mode (displays labeled data value)
- **IX**—Direct mode (input of value in x register)

In particular, one entry point, "IN", may be called whenever an application program must query the user for a numeric input value. As such, it is a direct replacement for the PROMPT instruction organic to the calculator, but automatically prompts, verifies and stores the received value using an indirect address contained in register 00. The printer version of the subroutine will automatically label and print the final, verified data value recorded.
Subroutine "IN" uses register 00 as a data location pointer and automatically increments this register so that subsequent calls to the subroutine will result in sequential data manipulation. The application programmer must insure that register 00 contains a number equal to the storage register number prior to calling the subroutine. For example, if register 00 contains 17, "IN" will store the data in register 17. One of the major advantages of this routine is that the same subroutine may be used to verify and/or change the data previously recorded. Thus, separate edit routines are usually unnecessary. Pressing the R/S key without touching any other key leaves the value stored unchanged. Pressing "1" and "+" and then "R/S" adds one to the current stored value, and so on for other function keys. Entering a new string of digits results in that new value being stored.

An additional feature of this subroutine is the "verify" mode indicated by flag 05. Flag 05 is reserved for this purpose and is set "on" by a call to subroutine "VR"—another of the subroutines in this common set. The verify mode is intended for use by a novice or other user who wishes to verify every data value as it is keyed into the calculator. The advantage is increased accuracy and confidence in the result.
Subroutine "D2"

Subroutine "D2" is used to set up the index register for a program loop. This subroutine has two alternative entry points which when called, increment different index registers as follows:

D2--Establishes register 02 as the index
D1--Establishes register 01 as the index

This subroutine is intended for use with the "ISG" loop structure which has the effect most like that of a FORTRAN "DO LOOP." For example, to execute a loop 20 times:

```
... 20
LEQ "D1
LBL 00
.....
ISG 01
GTO 00
...
```

The advantage of this form of loop structure is that register 00 may be used within the loop for address calculations. The first time the loop is addressed the integer portion of the number in register 00 will be 1, the second time it will be 2 and so on. There is no need to truncate the fractional portion of the number because the HP41C ignores the fractional component of a number when calculating a register address. Use of index registers for address calculation makes indirect addressing practical.
Registers 01 and 02 should be reserved for use as index registers by the application programmer. In most cases these two registers should prove sufficient.

Subroutine "VR"

Subroutine "VR" is used as a general purpose calculator initialization routine. This subroutine has three alternative entry points which vary the amount of initialization performed as follows:

- **VR**—Sets the verify mode on, and the following:
- **WR**—Suppresses the audio tones, and the following:
- **WS**—Clears all memory
  - Sets the display for integers,
  - Assigns statistical registers,
  - Sets "zero" level for equality testing, and
  - Sets base address for indirect addressing.

In the printer version of this initialization routine, the subroutine prints a banner (usually the program name) which has been stored in the alpha register prior to calling the initialization subroutine.

If flag 13 is set prior to calling the initialization routine, then the calling program must have placed the number of data registers required to execute the program in the x register prior to calling the initialization routine. In this case, a check will automatically be performed using subroutine "SZ" described below.

It is recommended that all application programs provide an alternate entry point which bypasses the initialization
step. Then if a data error is encountered, the user may review the data entered into the calculator by simply pressing the return key once after every prompt. This procedure works because subroutine "IN" recalls the stored value prior to prompting the user. When the user presses the clear key, the alphabetic prompt is removed and the existing data value revealed.

Subroutine "SZ"

Subroutine "SZ" is used to test if sufficient numbers of data registers are available to run an application program. This subroutine may be either called directly or as part of the initialization routine described above.

Subroutine "ER"

Subroutine "ER" is called whenever the application program encounters an error—usually in the input data. A prompt is displayed and an audio tone sounded, provided flag 26 (the audio enable flag) has not been cleared by the initialization routine described in paragraph D.

Subroutine "SORT"

This subroutine is included to illustrate the use of a stack register table in the program comments, but it also represents a useful utility routine. The sorting algorithms
used is the shell sort [Ref. 6: pp. 84-95] which gives reasonably fast sorting times with a very small program size. All conventions such as base register in R04 and number of data points in R15 are assumed by this subroutine. A complete list of all such register assignments is listed at the end of the program listing.

Subroutines "PUT" and "GET"

These two small routines provide a useful capability to store and recall up to three integers between 0 and 999 in one data register. This means that if you are manipulating a spread sheet of small, positive integers you can store the same data in one third the space. Of course, run time is degraded (about 20 seconds for every 100 data references.)

To store a value, assuming the base register has been defined, just press:

value ENTER point-number XEQ "PUT"

To recall a value, simply press:

point-number XEQ "GET"
7.7

COMMUNICATION ROUTINES

(( ))

IN

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

1 LBL "IN" (INPUT MODE--DISPLAY LABEL AND ?
2 SF 10 (SET QUERY ONCE FLAG
3 GTO 05 (OUTPUT MODE--DISPLAY LABEL AND DATA
4 LBL "IC" (OUTPUT MODE--DISPLAY LABEL AND DATA
5 CF 10 (INSURE NO QUERY
6 LBL 05 (ASSUMES ROO POINTS TO STORAGE REG
7 RCL IND 00 (DIRECT MODE--ASSUMES X REG HOLDS DATA
8 LBL "IX" (DIRECT MODE--ASSUMES X REG HOLDS DATA
9 ASTC 05 (ASSUMES LABEL SET UP BY CALLING PRGG
10 LBL 01 (QUERY OR DISPLAY VALUE?
11 FS? 10 (QUERY OR DISPLAY VALUE?
12 "M"? (QUERY OR DISPLAY VALUE?
13 CF 10 (QUERY OR DISPLAY VALUE?
14 FC? 10 (QUERY OR DISPLAY VALUE?
15 ARCL IND 00 (QUERY OR DISPLAY VALUE?
16 CF 22 (QUERY OR DISPLAY VALUE?
17 FC? 10 (QUERY OR DISPLAY VALUE?
18 GTO 05 (QUERY OR DISPLAY VALUE?
19 "-?" (QUERY OR DISPLAY VALUE?
20 CLA (QUERY OR DISPLAY VALUE?
21 FS? 22 (QUERY OR DISPLAY VALUE?
22 "-?" (QUERY OR DISPLAY VALUE?
23 GTO 02 (QUERY OR DISPLAY VALUE?
24 CF 10 (QUERY OR DISPLAY VALUE?
25 LBL 02 (QUERY OR DISPLAY VALUE?
26 LBL 02 (QUERY OR DISPLAY VALUE?
27 CF 10 (QUERY OR DISPLAY VALUE?
28 ARCL 05 (QUERY OR DISPLAY VALUE?
29 GTO 01 (QUERY OR DISPLAY VALUE?
30 LBL 03 (QUERY OR DISPLAY VALUE?
31 CF 03 (QUERY OR DISPLAY VALUE?
32 LBL 03 (QUERY OR DISPLAY VALUE?
33 GTO 01 (QUERY OR DISPLAY VALUE?
34 LBL 04 (QUERY OR DISPLAY VALUE?
35 GTO 04 (QUERY OR DISPLAY VALUE?
36 ARCL 05 (QUERY OR DISPLAY VALUE?
37 LBL 00 (QUERY OR DISPLAY VALUE?
38 RTN (QUERY OR DISPLAY VALUE?
39 LBL 05 (PRINT THE FINAL VALUE
40 ARCL 00 (PRINT THE FINAL VALUE
41 GTO 05 (PRINT THE FINAL VALUE

116
HP-41C SOURCE CODE: COMMON SUBROUTINES

36 LdL "O2  (SETUP LOOP USING REG 2 AS INDEX COUNTER
40 2
41 GTC 07  (SETUP LOOP USING REG 1 AS INDEX COUNTER
42 LdL "O1
43 1
44 LdL 07
45 X>Y  (NUMBER LOOP ITERATIONS MUST BE IN X
46 LE3  (PRIOR TO CALLING THIS SUBROUTINE.
47 /
48 1
49 +
50 STC IND Y
51 RTN
HP41C SOURCE CODE: COMMON SUBROUTINES

```

I-P4LC
SOURCE
COQ:
COMMON SUBRoutines

(-----------------------)
{                     }
{                     }
{                     }
----------------------

52 LSL "VR
53 SF C5
54 LBL "WR
55 CF 26
56 LBL "S
57 CLRG
58 GREG 10
59 CF 29
60 IE-C4
61 STC 03
62 13
63 STD 04
64 1
65 +
66 STO 00
(PRRT: ADV
(PRRT: SF 12
(PRRT: FC? 55
(PRRT: PRA
(PRRT: CF 12
(PRRT: ADV

67 FC? 13
68 RTN
69 X<> Z
```

(set verify mode on
(initialize for classroom use -- no audio
(standard initialization routine follows
(sets display--no decimal point if integer
(effective zero level
(normal indirect address base register.
(this is 1 less than the number of the
(first register where data is stored).
(double wide printing
(printer attached?
(set back to single wide print
(has initialization routine been asked to
(verify size?)

118
HP41C SOURCE CODE:  COMMON SUBROUTINES

((---------------------------------------------------)
)

((---------------------------------------------------)

)

((---------------------------------------------------)

)

70 LBL "SZ
71 "SET SZ=
72 ARCL A
73 1
74 SF 25
75 SF 25
76 RCL IND X
77 FC?C 25
78 PRCMPT
79 RTN

(PREPARE TO IGNORE ERROR

(SET FLAG 25 TO SEE IF SUFFICIENT

(SIZE EXISTS.

((---------------------------------------------------)

)

((---------------------------------------------------)

)

((---------------------------------------------------)

)

80 LBL "ER
81 0
82 LN
83 TONE 2
84 RTN

(DISPLAY "DATA ERROR" PRCMPT & SOUND TONE.

(BEST WAY TO DETERMINE WHERE ERROR

(OCCURRED IS TO HIT THE SST KEY ONCE,

(THEN GO INTO PRGM MODE.

119
SORT USES THE FOLLOWING REGISTERS:

R01 -- A
R02 -- B
R03 -- C
R04 -- INDIRECT ADDRESS BASE
R15 -- NUMBER DATA POINTS (SET BY &+1)
FP4IC SOURCE CODE: COMMON SUBROUTINES

{{{ SC }}

133 LBL "SC"
134 RCL 00
135 1
136 +
137 STC 05
138 1EC3
139 STC 10
140 1
141 3
142 /
143 STC 09
FP41C SOURCE CODE: COMMON SUBROUTINES

```
LBL "SA"

INT  STC 07
LASTX
RCL 09 / 12
+STO C3
RCL 07
RCL IND 07
INT
STC 14
LASTX
- RCL 10
INT
STO 13
LASTX
- RCL 10
* INT
STC 12
RCL 11
RCL IND 08
STO 11
FCYC 02
RTN
X<Y
STC IND 08
RCL 12
RCL 10
CHS
/ RCL 13
+ RCL 10
CHS
/ RCL 14
+ STC IND 07
RTN
```
HP41C SOURCE CODE: COMMON SUBROUTINES

THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG ASSIGNMENTS MADE BY THIS PROGRAM:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R00</td>
<td>Indirect address for storage of input data</td>
</tr>
<tr>
<td>R01</td>
<td>Loop index counter</td>
</tr>
<tr>
<td>R02</td>
<td>Loop index counter</td>
</tr>
<tr>
<td>R03</td>
<td>Effective zero level -- Use as temp if NA</td>
</tr>
<tr>
<td>R04</td>
<td>Base register for indirect addressing (15)</td>
</tr>
<tr>
<td>R05</td>
<td>Temp register for alpha prompt</td>
</tr>
<tr>
<td>R06</td>
<td>R09 = Application program temp registers</td>
</tr>
<tr>
<td>R10</td>
<td>R15 = Statistical registers -- Use as temp if NA</td>
</tr>
<tr>
<td>R16...</td>
<td>Storage of data via indirect addressing</td>
</tr>
</tbody>
</table>

FLAGS

F00-F0+ = Subroutine execution flags. Because these flags are visible in the display they can be set when entering a major subroutine and cleared when leaving -- giving the user a visual indication of what is happening inside the calculator. Use as temporary flags if this is not required.

F05 = Verify input mode. "On" when set. When set after every data value is entered, the calculator will echo the prompt and data value entered. If value is correct, simply press R/S key, otherwise, enter a corrected value and calculator will again ask for verification.

F10 = Used as a temporary flag inside "IC". Indicates no query prompt is desired.

F11 = Automatic execution flag -- Don't use ever

F12 = Double wide printing -- Local within "IC".

F13 = When set means main routine asking "VR" for automatic size check after initialization.

Application program may use freely after init.

F14 = F20 = Available for application program use.

202 END
<table>
<thead>
<tr>
<th>COMMON ROUTINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>COMMON SUBROUTINES</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
</tbody>
</table>
APPENDIX D

THE CROSS COMPILER PROGRAM AND COMMAND PROCESSOR

DESIGN METHODOLOGY:

This appendix discusses the design methodology used during construction of both the cross compiler program and the command processor, which is an IBM EXEC II program which provides an interactive programming environment for users of the system.

Blazie's compiler for the HP65 calculator [Ref. 9] represents one of the first attempts to provide a compiler for calculator programs. Both Carvalho [Ref. 10: pp. 25-29] and McNeal [Ref. 11: pp. 148-178] have published BASIC language programs which cross compile HP41CV instructions on a microcomputer for output to a line printer which can print acceptable bar code. While these referenced programs provided valuable insights into the problem, especially into the special characteristics of the HP41CV instruction set, none was exactly suited to the needs of this study. Because the Versatec plotter at the Naval Postgraduate School could be easily used only by FORTRAN programs, FORTRAN seemed the computer language of choice for this project. Both programs
were written with limited objectives and neither would have easily supported the extensions desired. Extensions planned for implementation included:

- An extended instruction set.
- In code comments.
- Extensive error checking.
- Compatibility with the Emulator.
- Synthetic Instructions [Ref. 1].
- Instruction macro's.

Having decided to code an original cross compiler, a design methodology which would capitalize on the advantages of FORTRAN was planned. FORTRAN's major deficiency for use in constructing a compiler of any type is its lack of alpha-numeric string handling capabilities. Rather than struggle with the lack of string functions, it was decided to code the necessary string functions as separate subroutines. This decision was reinforced countless times throughout the process of writing the compiler. The string function subroutines have been used not only in the cross compiler, but in many other FORTRAN programs since they were originally written. In fact, many persons who have no
interest in the HP41CV cross compiler may find the set of string functions listed in this thesis to be a valuable set of utility routines to be used to augment FORTRAN. The general convention used throughout the string function subroutines is that an alphabetic string may be represented as a vector of two byte integer variables used to store the characters and a single four byte integer variable used to store the length of the string.

One of the major advantages of the cross compiler is its ability to handle comments integrated within the HP41CV source code. This feature is critical to the clarification of the logical structure of the HP41CV programs. Because the parenthesis is not a valid HP41CV character, it was chosen as the comment indicator character. A comment may occur beginning at the first column on an input line or anywhere after an HP41CV instruction. The comment must follow the instruction because everything after the comment mark out to the end of the input line is considered part of the comment.

The control the user has over the output listing is also one of the advantages of the cross compiler. When two comment indicator marks are placed in positions one and two of the input line, the compiler will force a page eject when printing the output listing. In addition, the user can vary
the number of output lines per page and cause useful banners to be placed adjacent to program labels for ease of recognition.

Altogether there are twenty-four subroutines and a main program which constitute the cross compiler. The source code for each of these routines is provided in the second section of this appendix. Each subroutine begins with a statement concerning its function and construction. Accordingly, no general description of each subroutine will be repeated here. However, subroutine COMP deserves special attention, for it is the master lexicographic analyzer for the compiler and would also interface the user with the emulator. Its function is to receive a single line of HP41CV source code and identify it. COMP considers all HP41CV instructions to be of one of three types. The first category are the single byte instructions with no operands that can be compiled by a simple table look up. COMP has been constructed so that the instruction set can be extended at any time simply by increasing the size of this table. In this way abbreviated or altered command names could be easily used. The second category of instructions are the multi-byte instructions which require a table lookup and the translation of one or more operands, including possibly an
indirect instruction indicator. The table examined by the compiler is the same as for the category one instructions, and a code is given in the table which indicates to the compiler the number of operands which are required with each instruction. A syntax check is then made in subroutines IONE and ITWO to insure that the number and characteristics of each operand are appropriate for the given instruction. One of the major advantages of the use of the cross compiler is the syntax and error checking that is performed during the compilation process. The third type of instruction represents the exceptional instructions that are so difficult to compile that they require separate subroutines for efficient compilation. These instructions include storage and recall of data, program labels and program flow control statements such as goto and execute.

In order to provide an efficient programming command system for the compiler that would minimize the need to know technical details about the operation of the compiler, an IBM EXEC II program was written. This program not only interfaces the user to the compiler, but it also provides on line user instructions as to how to use the system. Included in this command processor is a command menu which gives the format and short description for each command.
Another command, help, provides more detailed information about each command. When a novice user first enters the exec, or types the name of the exec followed by a question mark, then he receives a four-page narrative description of what the system is, how it works, and what actions he must take to write a successful HP41CV application program.
$TRACE

******************************************************************************

*** HP41CV CROSS COMPILER COMMAND PROCESSOR  
***
*** THIS IBM EXEC II PROGRAM PROVIDES AN INTERACTIVE  
*** PROGRAMMING ENVIRONMENT FOR THE CONSTRUCTION OF  
*** HP41CV CALCULATOR PROGRAMS.  
***
*** WITH THE EXCEPTION OF THIS PROGRAM AND THREE OTHERS, ALL OF  
*** THE SOFTWARE IN THE HP41CV SYSTEM IS DESIGNED TO BE  
*** TRANSPORTABLE TO OTHER COMPUTER SYSTEMS WITHOUT EXTENSIVE  
*** PROGRAM MODIFICATION. THE INSTALLATION UNIQUE COMPONENTS  
*** ARE IN THE FOLLOWING ROUTINES:  
***
*** -- HP41CV EXEC  (THE COMMAND ENVIRONMENT)  
*** -- VERSA FORTRAN  (PLOTTING SUBROUTINE)  
*** -- PLOTPARM JCL  (PLOT CONTROL JCL)  
*** -- LBL XEDIT  (EDIT MACRO FOR LOWER CASE LABELS)  
***
*** ANOTHER VERSION OF THIS EXEC FOR USE WITH ASCII TERMINALS  
*** HAS BEEN PROVIDED. THIS ASCII ORIENTED EXEC MAY BE USED  
*** BY ENTERING THE COMMAND "HP41C". THE PRIMARY DIFFERENCES  
*** BETWEEN THESE TWO EXEC'S IS THAT FOR ASCII TERMINALS THE  
*** PRINTING OF THE COMMAND MENU IS SURPRESSED AFTER ONE PRINT  
*** AND COMMANDS WHICH HAVE MEANING ONLY FOR VIDEO TERMINALS  
*** SUCH AS FLIST, BROWSE AND XEDIT HAVE BEEN CHANGED TO THE  
*** CORRESPONDING TYPEWRITER TERMINAL EQUIVALENTS SUCH AS  
*** LISTFILE, TYPE AND EDIT.  
***
*** FOR THE NEW USER OF THE SYSTEM, IT IS RECOMMENDED THAT  
*** THIS PROGRAM BE EXECUTED SIMPLY BY TYPING THE COMMAND  
***
*** HP41CV  
***
*** FOR EXPERIENCED USERS, WHO HAVE NO NEED FOR THE DESCRIPTIVE  
*** INSTRUCTIONS, THE FOLLOWING COMMAND IS RECOMMENDED:  
***
*** HP41CV (FN) (1ST COMMAND)  

******************************************************************************

&IF /&2 = /HP41 &GOTO -CALLER
&IF /&2 = /&STACK &2
&IF /&3 = /&STACK &3
-CALLER
&IF /&4 = /&STACK &4
&IF /&5 = /&STACK &5
**CP SET PF01 IMMEL PF13**
**CP SET PF02 IMMEL PF14**
**CP SET PF03 IMMEL PF15**
**CP SET PF04 IMMEL PF16**
**CP SET PF05 IMMEL PF17**
**CP SET PF06 IMMEL PF18**
**CP SET PF07 IMMEL PF19**
**CP SET PF08 IMMEL PF20**
**CP SET PF09 IMMEL PF21**
**CP SET PF10 IMMEL PF22**
**CP SET PF11 IMMEL PF23**
**CP SET PF12 IMMEL PF24**
**CP SET PF13 IMMEL PF25**
**CP SET PF14 IMMEL PF26**
**CP SET PF15 IMMEL PF27**
**CP SET PF16 IMMEL PF28**
**CP SET PF17 IMMEL PF29**
**CP SET PF18 IMMEL PF30**
**CP SET PF19 IMMEL PF31**
**CP SET PF20 IMMEL PF32**
**CP SET PF21 IMMEL PF33**
**CP SET PF22 IMMEL PF34**
**CP SET PF23 IMMEL PF35**
**CP SET PF24 IMMEL PF36**
**CP TERMINAL LINEND OFF**
**CP SPDOL PRINTER CONT**

```
GLOBAL TXTLIB NONIMSL CMSLIB FORTMOD2 MCD2EEH IMSLSP
STATE INSTR CODES A
&IF ERET CODE -- 0 EUSERMODE = R
&IF EINDEX GT 0 &GOTO -PROGRAM
CLRSCREEN
&EGTYPE -ENDINTRO *** HP41C CROSS COMPILER ***
```

**THIS PROGRAM IS USED TO MAKE IT EASIER TO WRITE, DOCUMENT AND REVISE PROGRAMS FOR THE HP41C CALCULATOR. AS OUTPUT, THIS PROGRAM WILL PRODUCE OPTICAL BAR CODE FOR DIRECT ENTRY OF YOUR PROGRAM INTO AN HP41C OR HP41CV CALCULATOR.**

**WARNING ** THIS PROGRAM IS PART OF AN ONGOING RESEARCH PROJECT AND AS SUCH IS SUBJECT TO CONSTANT REVISION. **WHILE THERE ARE NO KNOWN ERRORS, THE PROGRAM HAS NOT BEEN EXTENSIVELY TESTED. TO INSURE THAT ANY ERRORS YOU DETECT ARE PROMPTLY CORRECTED. IT IS IMPORTANT THAT YOU SUBMIT AN ERROR REPORT TO THE PROGRAM PROponent AS SOON AS POSSIBLE.**
In order to go from a program in your head to the finished bar code, there are three main steps:

1. **Edit.** The program must be prepared as input to the cross compiler. The easiest way to do this is with the CMS EDIT facility.

2. **Compile.** The program must be processed by the cross-compiler. The cross-compiler is actually a FORTRAN program which produces two CMS files as output. Both these files have the same name as your program name, but have different file types. The "listing" file shows the results of the compile step including any errors, and the "data" file is a file of zeros and ones used by the bar code generator.

3. **Bar.** The "data" file from the compile step is used as input to produce the actual bar code. You should never permit this step until your program has successfully compiled without errors. This step is done by the batch processor and may take several hours to get your finished bar code.

Execution of the three steps necessary to produce bar code is under your control by selection of the appropriate step from a menu of commands which will appear at your terminal shortly.

The first step in using the cross-compiler is to prepare the source code (your program) on CMS disk. The first line of a source code file must contain the title of the program that is to be used as a label on the top of the bar code. This title should have no more than 40 letters. To help you remember that the label of the program must be the first line, you may receive a prompt asking you to enter the title when you first declare a new HP41 program. After you enter the title, your terminal will immediately shift to the CMS editor and you will see the title as the first line of the new file. This is your cue to enter the HP41C program that you have written. When you execute a "file" command in the editor mode, the terminal will display the command menu. You may then select to cross-compile the new program or any other option.
WHEN PREPARING YOUR SOURCE CODE PLEASE NOTE THAT LOWER CASE LETTERS ARE NOT THE SAME AS CAPITALS AND IN MOST CASES LOWER CASE WILL NOT BE ACCEPTED. IN ORDER TO MAKE IT EASY TO ENTER THE LOWER CASE ALPHABETIC LABELS, AN XEDIT MACRO "LBL" HAS BEEN PROVIDED. TO USE THIS MACRO, SIMPLY TYPE IN THE XEDIT COMMAND LINE, FOR EXAMPLE:

```
LBL LOWER A (FOR LOWER CASE "A" LABEL)
```

NOTE THAT THIS XEDIT MACRO ALSO DOES OTHER HELPFUL THINGS SUCH AS PROVIDING A BANNER TO HELP LOCATE LABELS AND DIRECTING THE CROSS-COMPILER TO START A NEW PAGE (INDICATED BY "11" IN COLUMNS 1 AND 2.) TO AVOID GOING TO A NEW PAGE WHEN YOU WRITE A LABEL, TYPE THE OPTION "NOPAGE" AS FOLLOWS:

```
LBL DOG NOPAGE (FOR AN ALPHA LABEL "DOG")
```

IN THE FUTURE, YOU MAY FIND IT MORE CONVENIENT TO SKIP THESE INSTRUCTIONS AND GO DIRECTLY TO THE "MENU" OF COMMANDS. TO DO THIS SIMPLY TYPE THE NAME OF THE CMS FILE WHICH CONTAINS OR WILL CONTAIN YOUR HP41C SOURCE CODE INSTRUCTIONS. AFTER THE INVOKING COMMAND "HP41C", AN EASY WAY TO DO THIS IS TO USE THE CMS "FLIST" FACILITIY FROM "FLIST" SIMPLY TYPE "PF19" IN THE COMMAND AREA.

NOW TO BEGIN:

```
-ENDINTRO

*** ESTABLISH A NEW HP41C PROGRAM SOURCE FILE. INCLUDES TITLE ***
*** PROMPTING. ***

-BEGIN

GET PROGRAM NAME (PF13 OR "STOP" RETURN TO CMS)
-ENDGET

GET PROGRAM CONTENTS... (PF13 OR "STOP" RETURN TO CMS)

GET PROGRAM NAME... (PF13 OR "STOP" RETURN TO CMS)
```

137
&NAME = &I
&TYPE = HP41
&MODE = A1
&SWITCH = OFF
STATE &NAME &TYPE &MODE
&IF &RET = 0 &GOTO -DISPLAY
CLRSCRN
&TYPE = ENDINTRO2

ENTER THE LABEL YOU WISH TO HAVE PRINTED AT THE TOP OF THE BAR CODE.
-INTRO2
&READ ARG
&STACK 1 61 62 63 64 65 66 67 68 69 10
&STACK 3 4 5 6 7
&STACK SET TRUNC 57
&STACK 1
XEDIT &NAME &TYPE &MODE

******************************************************************************
**
** COMMAND DISPLAY ROUTINE
**
**
******************************************************************************

-DISPLAY
&IF &SWITCH = /ON &GOTO -NEW
&IF &ASCII = /YES &GOTO -ENDISP
CLRSCRN
&TYPE = HP41 CROSS COMPILER ........ &NAME ........ EDITION=17 SEP 81
&TYPE = ENDISP
SELECT DESIRED COMMAND FROM THE FOLLOWING:

PF-KEY  COMMAND  CODE  ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM
PF13   STOP    S    GETS YOU OUT OF THE HP41 CROSS COMPILER
PF14   HELP    H    SHORT EXPLANATION OF HOW TO USE THE CROSS COMPILER
PF15   ENTER   E    INTERACTIVE PROGRAM ENTRY (NO FILE CREATED)
PF16   BAR      B    SUBMIT JOB FOR PHYSICAL PRODUCTION OF BAR CODE
PF17   NEW      N    BEGIN WORK ON A NEW PROGRAM OR NAMED SUBROUTINE
PF18   DIREC    D    DIRECTORY OF COMMANDS
PF19   LIST     L    DISPLAY NAMES OF HP41 PROGRAMS ON DISK
PF20   OLCOMP   C,O  OFFLINE COMPIL COMPILE AND AUTO GENERATION OF BAR CODE
PF21   PRINT    P    PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM
PF22   XEDIT    C    compiled SOURCE LISTING ON CMS DISK
PF23   COMP     C    EDIT THE PROGRAM USING THE CMS FULL-SCREEN EDITOR
PF24   ERASE    X    ERASE THE SOURCE FILE, LISTING FILE AND TEXT FILE
PF25   CMS      X    ALL EXECUTION OF ANY VALID CMS COMMAND
CP

ALLOWS EXECUTION OF ANY VALID CP COMMAND

-ENDDISP
-LINE INPUT COMMAND:
&READ ARGS
CLRSRCH

******************************************************************************
**
**  COMMAND CHECK ROUTINE
**
**
******************************************************************************

-CHECK

IF /CMS = /61 &GOTO -CMSCMD
IF /CP = /61 &GOTO -CPCMD
IF /PF13 = /61 &GOTO -EXIT
IF /PF14 = /61 &GOTO -HELP
IF /PF15 = /61 &GOTO -ENTER
IF /PF16 = /61 &GOTO -SUBMIT
IF /PF17 = /61 &GOTO -NEW
IF /PF18 = /61 &GOTO -DISPLAY
IF /PF19 = /61 &GOTO -LISTFILE
IF /PF20 = /61 &GOTO -OCOMP
IF /PF21 = /61 &GOTO -TYPE
IF /PF22 = /61 &GOTO -NOTYET
IF /PF23 = /61 &GOTO -COMPILE
IF /PF24 = /61 &GOTO -XEDIT
IF /STOP = /61 &GOTO -EXIT
IF /HELP = /61 &GOTO -HELP
IF /ENTER = /61 &GOTO -ENTER
IF /BAR = /61 &GOTO -SUBMIT
IF /NEW = /61 &GOTO -NEW
IF /DIREC = /61 &GOTO -DISPLAY
IF /LIST = /61 &GOTO -LISTFILE
IF /OCOMP = /61 &GOTO -OCOMP
IF /PRINT = /61 &GOTO -TYPE
IF /COMP = /61 &GOTO -COMPILE
IF /XEDIT = /61 &GOTO -XEDIT
IF /ERASE = /61 &GOTO -ERASE
IF /S = /61 &GOTO -EXIT
IF /H = /61 &GOTO -HELP
IF /E = /61 &GOTO -ENTER
IF /N = /61 &GOTO -NEW
IF /O = /61 &GOTO -DISPLAY
IF /p = /61 &GOTO -TYPE
IF /C = /61 &GOTO -OCOMP
IF /c = /61 &GOTO -COMPILE
IF /x = /61 &GOTO -XEDIT
IF /g = /FILEDEF &GOTO -INNER
GIF /G1 = /DATA &GOTO -INNER
GIF /ESWITCH = /ON &GOTO -PROGRAM
&TYPE 27 &L 62 63 64 65 UNRECOGNIZED
-ENDCHECK &GOTO -ENDDISP

******************************************************************************

*** PROCESS CMS OR CP COMMANDS PASSED TO THIS EXEC.  

******************************************************************************

-CMSMDO &ARG = &RANGE OF & 2 &N
&TRACE CN
&COMMAND &ARG
&TRACE OFF
&GOTO -ENDDISP
-CPCMD &ARG = &RANGE OF & 2 &N
&TRACE CN
&CP &ARG
&TRACE OFF
&GOTO -ENDDISP
-XEDIT
ESTACK SET TABS 1 20 25 35 45 55 65
ESTACK SET TRUNC 57
XEDIT &PROGNAME &PROGTYPE &PROGMODE
&GOTO -DISPLAY

******************************************************************************

** LIST HP41C PROGRAM FILES ON CMS DISK **

******************************************************************************

-LISTFILE
FLIST * HP41 A
&GOTO -DISPLAY

******************************************************************************

** COMPIL **

******************************************************************************

-COMPIL
FILEDEF 05 DISK &PROGNAME &PROGTYPE
FILEDEF 06 DISK &PROGNAME LISTING
FILEDEF 04 DISK &PROGNAME DATA
FILEDEF 02 DISK INSTR CODES &USERMODE
&TYPE CROSS-COMPIL BEGINS.
HPCCROSS
BROWSE &PROGNAME LISTING
&GOTO -DISPLAY

******************************************************************************
OFFLINE COMPILE

***********************************************************************

-OCCMP
-ERASE &PRGNAME DATA
-FILEDEF 05 DISK &PRGNAME &PROTYPE
-FILEDEF 06 DISK &PRGNAME LISTING
-FILEDEF 04 DISK &PRGNAME DATA
-FILEDEF 02 DISK INSTR CODES &USERMODE
&TYPE CROSS-COMPILE BEGINS.. HPCROSS
PRINT &PRGNAME LISTING (UP
STATE &PRGNAME DATA
ZIF &RETENCE = 0 &GOTO -SUBMIT
&TYPE COMPILATION OF &PRGNAME WAS NOT SUCCESSFUL.
&GOTO -ENDDISP
***********************************************************************

*** USING THE INTERACTIVE MODE, ENTER A NEW PROGRAM. ***

***********************************************************************

---ENTER

&GOTO -ENDCOAUCTION

USE OF THE INTERACTIVE ENTRY MODE REQUIRES THAT YOU PROPERLY
CONTROL THE USE OF UPPER AND LOWER CASE. ALSO, INTERACTIVE
ENTRY DOES NOT CREATE A PERMANENT RECORD OF YOUR SOURCE CODE
INPUT. SHOULD A REVISION BE REQUIRED, YOU WOULD NEED TO
RE-ENTER THE COMPLETE PROGRAM, FOR THESE REASONS YOU MAY WISH
TO EDIT A SOURCE CODE FILE FIRST, AND THEN SUBMIT THIS FILE
FOR CROSS-COMPILED WITH THE "COMP" COMMAND.

DO YOU WISH TO PROCEED WITH INTERACTIVE ENTRY? (Y/N)

INPUT RESPONSE:
-ENDCOAUCTION
-GREAT ARG
E IF /D1 = /Y &GOTO -DISPLAY
C1=CRScRN
&TYPE FIRST ENTER THE LABEL YOU WISH TO BE PRINTED AT THE TOP OF THE
&TYPE BAR CODE;
&TYPE THEN ENTER THE INSTRUCTIONS IN YOUR PROGRAM (IN UPPER CASE EXCEPT
&TYPE FOR LOWER CASE ALPHABETIC LABELS WHICH ARE ALLOWED.)
&TYPE INPUT:
-FILEDEF 04 DISK &PRGNAME DATA
-FILEDEF 02 DISK INSTR CODES &USERMODE
HPCROSS
&GOTO -ENDDISP
*** DISPLAY A MESSAGE THAT FUNCTION IS NOT AVAILABLE. ***

-NOTYET
CLRSCRN
&BEGIN -ENDYET

THE FUNCTION YOU HAVE REQUESTED IS NOT YET AVAILABLE. IF YOU HAVE ANY IDEAS THAT SHOULD BE INCLUDED HERE, PLEASE CONTACT THE PROGENET.

THANK YOU.
-ENDYET
&GOTO -ENDDISP

******************************************************************************

******************************************************************************

** TYPE LISTING FILE **

******************************************************************************

******************************************************************************

** ERASE SOURCE, LISTING AND TEXT FILES **

******************************************************************************

******************************************************************************

-ERASE
ERASE &PRGNAME LISTING
ERASE &PRGNAME DATA
ERASE LCLA MAP
&TYPES WARNING - DO YOU WISH TO ERASE THE SOURCE CODE? YES/NO?
&READ VARS &ANSW
&IF &ANSW = /YES &GOTO -DISPLAY
ERASE &PRGNAME &PROTYPE &PROGMODE
&GOTO -DISPLAY

******************************************************************************

******************************************************************************

** SUBMIT TO MVS FOR BATCH PROCESSING **

******************************************************************************

******************************************************************************

-SUBMIT
-TRYSUBMIT
&PERM - SUBMIT
&BEGIN -ENDSUBM
ENTER JOB NAME AND USERID:
-ENDSUBM
&READ VARS &JN &USERID
&JN = &PIECE OF &JN 1 8
&USERID = &PIECE OF &USERID 1 4
&STACK I J &JN JOB &USERID J01111, 'HP41CV BAR CODE', CLASS=A
&STACK C //10111//, 10111/
&STACK I // EXEC FRXCLGP
&STACK I // FORT.SYSIN DD *
&STACK GET VERSA FORTRAN &USERMODE
&STACK I //GO.PLOTPARM DO *
&STACK GET PLOTPARM JCL &USERMODE
&STACK I //GO.SYSIN DD *
&STACK GET &PROGNAME DATA
&STACK I /*
&STACK FILE
&EDIT &PROGNAME JCL
EXEC SUBMIT &PROGNAME JCL
ERASE &PROGNAME JCL
&GOTO -DISPLAY
**************************************************************************
**
** PRINT INSTRUCTIONS
**
**
**************************************************************************
-HELP
&BETYPE -ENDHELP

HP41C CROSS COMPILER COMMAND PROCESSOR

YOU ARE CURRENTLY EXECUTING A CMS EXEC FILE THAT MAKES IT EASY TO INVOKE THE HP41C CROSS COMPILER AND WRITE PROGRAMS USING CMS AND THE IBM 3278 DISPLAY TERMINAL. COMMON PROGRAMMING REQUIREMENTS SUCH AS EDITING CAN BE ACCOMPLISHED IN THREE WAYS:

--USING THE PROGRAMMED FUNCTION KEYS (PF KEYS)
--USING A SHORT COMMAND WORD
--USING A ONE OR TWO LETTER MNEMONIC CODE

THE COMMAND ACTIONS AND THEIR ASSOCIATED PF KEYS AND CODES ARE ALL GIVEN IN A DIRECTORY WHICH IS DISPLAYED WHEN THE COMMAND PROCESSOR IS WAITING FOR YOUR INPUT. MORE DETAILS ABOUT THE AVAILABLE COMMANDS FOLLOWS:

PF13 STOP S THIS COMMAND IS USED WHEN YOU WISH TO STOP PROCESSING
PF21 PRINT P
THIS COMMAND PRINTS A COPY OF THE "LISTING" FILE ON
THE HIGH SPEED PRINTER. IF YOU WISH TO HAVE A PRINTED
COPY OF THE SOURCE CODE WITHOUT THE CROSS-COMPILER'S
FEEDBACK, IT IS BEST TO SIMPLY PRINT THE SOURCE CODE
CMS FILE BY ISSUING THE CMS PRINT COMMAND.

PF22 GO G
THIS COMMAND IS USED TO INVOKE THE HP41C EMMULATOR
PROGRAM WHICH ALLOWS YOU TO TEST EXECUTION OF THE
PROGRAM ON THE LARGE COMPUTER. THE EMMULATION PROGRAM
WILL EXECUTE THE PROGRAM EXACTLY AS YOUR CALCULATOR
WOULD. THIS COMMAND HAS NOT BEEN IMPLEMENTED AS OF
17 SEP 81.

PF23 COMP C
THIS COMMAND IS USED TO INVOKE THE CROSS COMPILER TO
TRANSLATE AN HP41C PROGRAM WRITTEN ON CMS DISK IN
SOURCE CODE FORM. AFTER THE COMPILER THE USER IS
AUTOMATICALLY PLACED IN THE CMS BROWSE MODE FOR THE
OUTPUT "LISTING" FILE THAT RESULTED FROM THE COMPILE.

PF24 XEDIT X
THIS COMMAND IS USED TO INVOKE THE FULL-SCREEN EDITOR
TO MAKE MODIFICATIONS TO THE HP41C SOURCE CODE FILE.

IF YOU HAVE PROBLEMS USING THIS COMMAND PROCESSOR OR HAVE A SUGGESTION
FOR IMPROVEMENT, PLEASE CONTACT THE PROONENT FOR THE HP41C SYSTEM.

-ENHELP
&GOTC -ENMODISP
******************************************************************************
** EXIT COMMAND PROCESSOR **
******************************************************************************

-EXIT
CP SPOOL PRINTER CLOSE NOCONT
CLRSCRN
&EXIT
C***********************************************************************
C** THIS IS THE MAIN ROUTINE FOR THE HP14C CROSS-COMPILER.              
C** INPUT TO THIS PROGRAM IS A SUPERSSET OF HP14C INSTRUCTIONS. THE    
C** REGULAR INSTRUCTIONS ARE WELL DOCUMENTED IN THE HP14C OWNER'S    
C** HANDBOOK AND PROGRAMMING GUIDE [SEE ESPECIALLY THE INDEX ON   
C** PAGE 271]. IN ADDITION TO THESE REGULAR INSTRUCTIONS, THE     
C** FOLLOWING COMMANDS ARE SUPPORTED:                                
C** ( ) INDICATES A COMMENT CARD (NO INSTRUCTIONS GENERATED)        
C** XROM        SUBROUTINE CALL TO A READ ONLY MEMORY                  
C** LBL         KEY ASSIGNMENTS MAY BE MADE AS PART OF ALPHA LBL      
C** OUTPUT OF THIS PROGRAM IS AN INTERMEDIATE FILE OF ZERO'S AND     
C** ONE'S REPRESENTING HP14C MACHINE INSTRUCTIONS. THIS FILE WILL    
C** NORMALLY BE PASSED TO A FORTRAN PROGRAM WHICH DRAW HP14C BAR  
C** CODE ON A HIGH RESOLUTION PLOTTER.                              
C***********************************************************************

INTEGER ASGN$,EQ$,POS$,SEG$,PARS$,CON$,IN$,COMP$,FIND$,LCUT$
INTEGER TRIMS$
COMMON/TEXT/IDIM,IPRT
COMMON/FLAGS/DONE,PO
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
COMMON/LOG/DONE,PO
LOGICAL DONE,PO
INTEGER# P1,P2,P3,P4,P5,P6,P7,P8,S1,S2
COMMON/TABLE/INST$,LINST,CODE,NINST
INTEGER# INST$(111),CODE(111),NINST
INTEGER# TITLE$(133)
INTEGER# TITLE,TITLE,
INTEGER# Title,Func$,'MAIN'
INTEGER# COMENT/'*/
INTEGER# UNDER(15)/15*',
INTEGER# M(8000)/8000-997'
INTEGER# M(1/1.81/1
INTEGER# LINCNT/70/PAGE/72/
LOGICAL ERROR/.FALSE./
NUMPG=1
LITLE=-80
MITLE=1
IDIM=80
MAX=2000
NINST=111
PO
PALPHA=..FALSE.
DONE= .FALSE.

READ(2,101) (PRT,PAGE
READ(2,9101) (TITLE$(JJ),JJ=1,25)
FORMAT(25A1)

101 FORMAT(125)
DO 5 I=1,NINST
    READ(2,1021)(INST$(J,I),J=1,6),CCOE(I),LINST(I)
FORMAT(6A1,4X,I5,10X,5)
CONTINUE

5

READ(5,103) (TITLE$(I),I=26,1DIM)
FORMAT(75A1)

LENGTH=0
DO 16 I=1,MAX
    ( I = THE INSTR NUMBER, J = CHARACTER IN INSTR)

ATTEMPT TO READ A TEXT STRING.
    IF(IPRT.GE.20)WRITE(6,292) UNDER
292 FORMAT('I',*NEXT INSTRUCTION:','15A4,/
    IF(IN$(I$,LT,5)) 14,12,12

GO TO THE FOLLOWING INSTRUCTIONS IF A CHARACTER STRING FOUND

CHECK FOR A COMMENT CARD AND/OR PAGE EJECT

*** TWO "COMENT" CHARACTERS IN POSITION 1 AND 2 OF AN INPUT
LINE ARE CONSIDERED A MANDATORY PAGE EJECT PRAGMA. ***
I2 IF(T$(1).NE.COMENT)GOTO 13
   IF(MOD(LINCNT, PAGE).EQ.0.OR.(T$(2).EQ.COMENT.AND.LT.GE.2))
      IF(LINCNT+1.LT.PAGE)  CALL NEWPG$(LINCNT,NUMPG, TITLE$,LTITLE,1)
   LINCT=LINCNT+1
   WRITE(6,268) (T$(J),J=1,LT)
   FORMAT(110A1)
   IF(IPRT.GE.10)WRITE(6,263)
263 FORMAT(' FOUND COMMENT CARD. NOTHING MORE DONE.')
   GOTO 16
C
C IF NOT A COMMENT, INCREMENT THE INSTRUCTION COUNTER AND PRINT
C THE INSTRUCTION.
13 LENGTH=LENGTH+1
   IF(MOD(LINCNT, PAGE).EQ.0)
      IF(LINCNT+1.LT.PAGE)  CALL NEWPG$(LINCNT,NUMPG, TITLE$,LTITLE,1)
   LINCT=LINCNT+1
   WRITE(6,269)LENGTH,(T$(J),J=1,LT)
   FORMAT(110A1)
C
C TRIM OFF TRAILING COMMENTS
C IF(FIND$(COMENT,1,T$,LT,LOC)) 6000,9915,9914
9914 LT=LOC-1
9916 IF(TRIM$(T$,LT)) 6000,9916,9915
9915 CONTINUE
   GOTO 16
   CONTINUE
C
C COMPILE THE TEXT INSTRUCTION.
C IF(COMP$(T$,LT,M,ML)) 15,16,20
15 ERROR=.TRUE.
16 CONTINUE
C
C GOTO THE FOLLOWING INSTRUCTIONS IF END OF FILE ENCOUNTERED
14 WRITE(6,259)
259 FORMAT(' *****************************************END OF FILE*****************************************')
GOTO THE FOLLOWING INSTRUCTIONS IF END COMPILATION

IF( ERROR ) STOP

CALL THE BAR CODE GENERATOR.

M$AVE=M1
B1=1
IBAR=BSTR$(M,M1,ITOT,TITLE$)
WRITE(6,301) ITOT

FORMAT(' END HP41C CROSS COMPIL$E',15,' BYTES IN TOTAL PROGRAM')

STOP

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
FORMAT(' *** STRING LENGTH ERROR *** ',A4)
MAIN$=0
RETURN

WRITE(6,6003) FUNC$
FORMAT(' *** FUNCTION CALL ERROR *** ',A4)
MAIN$=0
RETURN
END
INTEGER FUNCTION A1NS(INOP, B)

** This function converts a decimal number <=256 into a binary number, with the values of the binary digits stored in successive elements of an 8 element array of integers.

** This function is called from BSTR$ in the HP-41CV compiler.

The return value of the function A1NS is set as follows:
- 0 = Continue to compile
- 1 = An error in compiling the instruction.

**

**

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDM, IPRT
INTEGER#4 FNC##1 A1NS/
INTEGER#0 B[8]
OPERND=INOP

CHECK FOR VALID ENTRY OPERAND

IF((OPERND.GT.255).OR.(OPERND.LT.0)) GOTO 6000

CONVERT THE FIRST BINARY DIGIT

D1=OPERND-128
IF(D1) 100,110,110
100 B(1)=0
GOTO 120
110 B(1)=1
GOTO 120
OPERND=D1

CONVERT THE SECOND BINARY DIGIT

D2=OPERND-64
IF(D2) 200,210,210
200 B(2)=0
GOTO 230
210   B(2) = 1
      OPERND = D2

CONVERT THE THIRD BINARY DIGIT

230   D3 = OPERND - 32
      IF (D3) 300, 310, 310

300   B(3) = 0
      GOTO 340

310   B(3) = 1
      OPERND = D3

CONVERT THE FOURTH BINARY DIGIT

340   D4 = OPERND - 16
      IF (D4) 400, 410, 410

400   B(4) = 0
      GOTO 450

410   B(4) = 1
      OPERND = D4

CONVERT THE FIFTH BINARY DIGIT

450   D5 = OPERND - 8
      IF (D5) 500, 510, 510

500   B(5) = 0
      GOTO 560

510   B(5) = 1
      OPERND = D5

CONVERT THE SIXTH BINARY DIGIT

560   D6 = OPERND - 4
      IF (D6) 600, 610, 610

600   B(6) = 0
      GOTO 670

610   B(6) = 1
      OPERND = D6

CONVERT THE SEVENTH BINARY DIGIT

670   D7 = OPERND - 2
      IF (D7) 700, 710, 710

A1N00400
A1N00500
A1N00510
A1N00520
A1N00530
A1N00540
A1N00550
A1N00560
A1N00570
A1N00580
A1N00590
A1N00600
A1N00610
A1N00627
A1N00630
A1N00640
A1N00650
A1N00660
A1N00670
A1N00680
A1N00690
A1N00700
A1N00710
A1N00720
A1N00730
A1N00740
A1N00750
A1N00760
A1N00770
A1N00780
A1N00790
A1N00800
A1N00810
A1N00820
A1N00830
A1N00840
A1N00850
A1N00860
A1N00870
A1N00880
A1N00890
A1N00900
A1N00910
A1N00920
A1N00930
A1N00940
A1N00950
A1N00960
700  B(7)=0
710  GOTO 780
       B(7)=1
       OPERNO=07
       C
       C
       CCNVET THE EIGHTH BINARY DIGIT
       C
780  D8=OPERNO-1
800  IF(D8) 800,810,6000
810  B(8)=0
     GOTO 1000
     B(8)=1
     C
     C
     WRITE OUT CONVERSION IF NECESSARY AND RETURN
     C
1000 IF(IPRT.GE.20) WRITE(6,66) INOPER,(B(I),I=1,8)
       66 FORMAT('TRACEx',AIN$ OPERAND='';I5;'BINARY=';8I1)
       AIN$=0
       RETURN
     C
     C
     ERROR HANDLING SECTION FOLLOWS
     C
6000 WRITE(6,6001) Func$  
6001 FORMAT('**** CONVERSION ERROR **** ','A4)
       WRITE(6,6002) INOPER
       6002 FORMAT('ERROR IN AIN$ OPERAND=';I5)
       AIN$=-1
       RETURN
     END
INTEGER FUNCTION ALPHS(A$,LA,M,I)
C******************************************************************************
C** THIS FUNCTION INTERPRETS ALPHABETIC CHARACTERS INTO HP41C KEY CODES.   
C** THE RETURN VALUE OF THE FUNCTION ALPHS IS SET AS FOLLOWS:     
C** 0 = CONTINUE TO COMPILE    
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION.  
C******************************************************************************
IMPLICIT INTEGER(A-Z)
COMMON/FLAGS/DONE,PG,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
LOGICAL DONE,PG,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
LOGICAL DONE,PG,DIGIT,ALPHA
INTEGER A$(IDIM),BLNK,'/QUOTE/""/
INTEGER*4 FUNC$,'ALPH$/
INTEGER*4 M(I)
INTEGER*4 FUNC$/'COMP '/
INTEGER*2 C$(60)/

153
DO 35 I=LA,1,LA
IF(A$(I).NE.QUOTE) GOTO 15
HAVE FOUND A QUOTE MARK--DISREGARD IF 1ST INSTR,ELSE CUIT
IF(I.EQ.1) GOTO 35
I2=FIND$4(A$(I),LA,C$,LC,LOC)
IF(I2.NE.0)GOTO 20
WRITE(6,207)M1
207 FORMAT(* **** INVALID CHARACTER *****",5X,15)
A$(I)=BLNK
35 CONTINUE
IZ=1
20 M(M1)=C2(IZ)
IF(IPRT.GE.10)WRITE(6,208)M1,C$(IZ),CZ(IZ),M(M1)
208 FORMAT('ALPH$','I5','ALPHA CHARACTER','I75','I3') 3X,A1,3X,I3
2 M1=M1+1
35 CONTINUE
C
40 ALPH$=0
RETURN
C
ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$ 6001 FORMAT('*** STRING LENGTH ERROR *** ','A4')
WRITE(6,6010) LA, LB, IDIM
6010 FORMAT('LA=', 'I10','LB=', 'I10','IDIM=', 'I10')
ALPH$=-1
RETURN
END
INTEGER FUNCTION ALPI$(A$,LA,M,M1)

** ALP00000
C** STRING A$ BEGINS WITH A QUOTE AND HENCE IS AN ALPHA ENTRY
C** INSTRUCTION. THIS ROUTINE Compiles SUCH INSTRUCTIONS.
C**
C** THE RETURN VALUE OF THE Function ALPI$ IS SET AS Follows:
C** 0 = CONTINUE TO Compile
C** -1 = AN ERROR IN Compiling THE Instruction.
C**
C**
C**************************************************************************************
C Implicit INTEGER(A-Z)
C COMMON/TEXT/IDIM,IPRT
C COMMON/FLAGS/DONE,POIGIT,PALP,iA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C COMMON/CNTR/P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C LOGICAL DONE,POIGIT,ALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C INTEGER*4 P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C INTEGER*2 A$(IDIM)
C INTEGER*2 BLNK$/'/
C INTEGER*2 QUOTE/**,APPEND/**
C INTEGER*4 Func$/**ALPI$/**
C INTEGER*4 M(i)
C IF(IPRT.GE.10) WRITE(6,200)LA,('A$(I),I=1,LA)
C FORMAT('TRACE '/(13,'4$ALPI$ I = ',10A1)
C IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C STRIP OFF THE LEADING QUOTE
C IF(LCUT$(A$,LA,1)) 6015,6015,10
C
C STRIP OFF THE TRAILING QUOTE, IF any
C IF(A$(LA).EQ.QUOTE)LA=LA-1
C
C SET THE LENGTH OF THE INSTRUCTION
C IF(LA.LT.15)GOTO 15
C WRITE(6,204)
C FORMAT('**** ALPHA STRING TOO LONG *****)
C LA=15
C ALP00480
15    IBYTE=IBYTE+1
     M(MI)=IBYTE
     IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215    FORMAT(' ALP$','15,' LENGTH OF THIS INSTR IS','13)
     M1=M1+1
     C
     C
     ENCODE THE TEXT LENGTH INSTRUCTIONS
     C
     M(MI)=240+LA
     IF(IPRT.GE.10)WRITE(6,211)M1,M(MI)
211    FORMAT(' ALP$','15,' TEXT LENGTH INSTR','T75','13)
     M1=M1+1
     C
     C
     CHECK FOR ALPHA APPEND INSTRUCTION
     C
     IF(A$(1).NE.APPEND)GOTO 50
     C
     HAVE IDENTIFIED AN ALPHA APPEND INSTRUCTION
     M(MI)=127
     IF(IPRT.GE.10)WRITE(6,214)M1,M(MI)
214    FORMAT(' ALP$','15,' ALPHA APPEND CHAR','T75','13)
     M1=M1+1
     IF(LCUT$(A$,LA,1)) 6015,6015,50
     C
     C
     ENCODE TEXTUAL STRING
     C
     50    ALP$=ALPH$(A$,LA,P,M1)
     RETURN
     C
     ERROR HANDLING SECTION FOLLOWS
     C
     6000  WRITE(6,6001) FUNC$
     6001  FORMAT(' *** STRING LENGTH ERROR *** ','A4)
     WRITE(6,6010) LA,LB,IDIM
     6010  FORMAT(' LA='','110',' ','LB='','110','IDIM='','110')
     ALP$=-1
     RETURN
     6015  WRITE(6,6016)
     6016  FORMAT(' **** INVALID OPERAND IN ALPHA ENTRY INSTR ****')
     ALP$=-1
     RETURN
     END
INTEGER FUNCTION ASGN$(A$, $LB$)

**FUNCTION** IS A STRING ASSIGNMENT OPERATOR. THE STRING IN A$ IS COPIED INTO B$. THE NULL STRING LA=0 IS A VALID STRING AND WILL BE COPIED CORRECTLY.

COMMON/TEXT/LA, LB, IPRT

INTEGER*2 A$(IDIM), B$(IDIM)

IF(IPRT.GE.10) WRITE(6, 200) LA, (A$(I), I = 1, LA)

200 FORMAT('TRACE ', (4, 'A'), 'ASGN$: ', 'A', 'I10A1)

IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

IF(LA.EQ.0) GOTO 20

DC 15 I = 1, LA

B$(I) = A$(I)

15 CONTINUE

20 LB = LA

ASGN$ = 1

RETURN

**ERROR HANDLING SECTION**

6000 WRITE(6, 6001) FUNC$

6001 FORMAT('STRING LENGTH ERROR *** ', 12I4)

WRITE(6, 610) LA, LB, IDIM

610 FORMAT('LA=' , 'I10', 'LB=' , 'I10', 'IDIM=' , 'I10')

ASGN$ = 0

RETURN

END
INTEGER FUNCTION BSTR$(M, M1, TOTAL, TITLE$)

C** THIS FUNCTION TAKES AN ARRAY (M) OF MACHINE CODE (DECIMAL)
C** INSTRUCTIONS AND CONVERTS THEM INTO AN ARRAY (W) OF BINARY
C** INSTRUCTIONS. IT ALSO COMPUTES THE BARCODE CHECKSUM, AND
C** SEGMENTS THE ARRAY INTO BARCODE LINES.
C**
C** THE RETURN VALUE OF THE FUNCTION BSTR$ IS SET AS FOLLOWS:
C** 0 = CONTINUE TO COMPILDE
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C**
IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 M(133), TITLE$(133)
INTEGER*2 ALPHA(133), BLNK/*, ZERO/0, ONE/1/
INTEGER*4 FUNC$, 'BSTR$'
INTEGER*4 M(I), M1, M1
IF(IPRT.GE.10) WRITE(6,200)
200 FCRMAT('TRACE', I3, 'BSTR$','

C C C
C INITIALIZE COUNTERS
C
CHECK=0
TOTAL=0
SEQNUM=0
LEAD=0
ROW=0
P=1
W1=27

C C C
C WRITE THE TITLE TO THE BINARY CODE ARRAY
C
WRITE(4,776)(TITLE$(JJ), JJ=26, IDIM)
776 FORMAT(80A1)

C C C
C CHECK FOR END OF PROGRAM
C
320 IF(M(P).LE.-99) GOTO 530
C  EXTRACT NUMBER OF BYTES IN INSTRUCTION
C    IBYTE=M(P)
C    NBYTE=IBYTE
C    P=P+1
C
C  EXTRACT NEXT OPERAND OF THE INSTRUCTION
C    OPERND=M(P)
C    P=P+1
C
C  CONVERT OPERAND TO BINARY AND LOAD INTO ARRAY W
C    CHECK=CHECK+OPERND
C    IF(CHECK.GT.255) CHECK=CHECK-255
C    IF(IPRT.GE.10) WRITE(6,555)ROW,OPERND,CHECK
C    555 FORMAT(*, SEND TO AINS ROW=**,13,** OPERAND=**,16,** CHECKSUM=**,15)
C    IF(AINS(OPERND,MI111)) 8000,420,420
C
C  IF SUCCESSFUL CONVERSION, DECREASE BYTES REMAINING
C  INCREMENT THE ROW COUNT, AND CHECK TO SEE IF END OF BARCODE ROW
C    IBYTE=IBYTE-1
C    W1=W1+8
C    ROW=ROW+1
C    IF(ROW.EQ.13) GOTO 530
C
C  CHECK TO SEE IF INSTRUCTION HAS BEEN COMPLETELY ENCODED
C    IF(IBYTE.EQ.0) GOTO 520
C    GOTO 390
C
C  PROCESS END OF BARCODE ROW, FIRST SAVE ENDING LOCATION IN TEMP
C  BARCODE ROW (THIS LOCATION WILL BE DIFFERENT DEPENDING ON
C  WHETHER YOU ENTER ROUTINE BY DETECTING END OF ROW CR BY END OF
PROGRAM, THEN CHECK FOR CONTINUATION OF INSTRUCTION THAT MUST CROss BARCODE BOUNDARIES.

530  \( WP=W_1 \)
      IF(18YTE,NE,0) GOTO 560
      GOTO 580

CALCULATE NUMBER OF TRAILING BYTES IN BARCODE ROW

560  TRAIL=NBYTE-1BYTE

COMPUTE THIRD BYTE OF BARCODE ROW AND CONVERT TO BINARY

580  \( THIRd=(16*LEAD)+TRAIL \)
      \( WI=19 \)
      CHECK=CHECK+THIRD
      IF(1PRT,GT,255) CHECK=CHECK-255
      IF(1PRt,GE,10) WRITE(6,551)ROW,THIRD,CHECK
      IF(IN$1THIRD,\( W(1)1) \) 6000,1090,1090

COMPUTE SECOND BYTE OF BARCODE ROW AND CONVERT TO BINARY

1090  \( SECD=(16+MOD(\textit{SEQNUM},16) \)
      SEQNUM=SEQNUM+1
      \( WI=11 \)
      CHECK=CHECK+SECD
      IF(1CHECK,GT,255) CHECK=CHECK-255
      IF(1PRt,GE,10) WRITE(6,555)ROW,SECD,CHECK
      IF(IN$1(SECND,\( W(1)1) \) 6000,1180,1180

C
COMPUTE FIRST BYTE OF BARCODE ROW AND CONVERT TO BINARY

THE FIRST BYTE CONTAINS THE CHECKSUM. THIS BYTE IS A PARITY CHECK IN THE FORM OF A RUNNING SUMMATION, MODULO 256 WITH A WRAP-AROUND CARRY (0, 1, 2,... 255, 256, 1, 2,...).

L80  W1=3
1 FIRST=CHECK
1 IF(IPRT.GE.10) WRITE(6,555)W1,FIRST,CHECK
1 IF(AIN$='FIRST',W(W1)) 6300,1220,1220

ADD THE START AND STOP BITS AND ADD AN END OF ROW FLAG

220  IF(IPRT.GE.20) WRITE(6,556)
556  FORMAT(' END OF BARCODE ROW$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ '
4 W1=0
4 W(2)=0
4 W(WP)=1
4 ENDING=W(ENDING)+1
4 W(ENDING)=0
4 ENDBIT=W(ENDING)+2
4 W(ENDBIT)=-99

TRANSFER THE COMPLETED BARCODE ROW EITHER DIRECTLY TO THE
1 PLOTTER, OR TO AN ARRAY OF INTEGER*2 VARIABLES WHICH HOLD
1 ZERO'S OR ONE'S

******************************************************************************

INSERT CALL TO VERSATEC HERE.

******************************************************************************

588  IF(IPRT.GE.20) WRITE(6,558) B1,ENDBIT
588  FORMAT(' TRANSFER TO BINARY ARRAY AT',I7,' NUMBER DIGIT $$
558  DO 1350 I=1,ENDING
558    IF(W(I)) GOTO 559
      ALPHAI=ZERO
      GOTO 1350
559    ALPHAI=ONE
      GOTO 1350
1350 CONTINUE
IF(ENDING.EQ.132) GOTO 1736
DO 1735 I=ENDBIT,132
ALPHA(I)=BLNK
1735 CONTINUE
WRITE(*,777) (ALPHA(I), I=1,132)
777 FORMAT(66A1,,5X,66A1)
IF(REQUIRED, PRINT THE BARCODE ROW AS ZERO'S AND ONE'S ON PAPER
IF(IPRT.GE.20) WRITE(6,201)(W(I), I=1,ENDING)
201 FORMAT(*,**,13211)

SET NUMBER OF LEADING BYTES FOR NEXT ROW AND RE-INITIALIZE
LEAD=IBYTE
TOTAL=TOTAL+ROW
ROW=0
W=27
IF(M(P)) 1400,480,480

SET FINAL VALUES AND RETURN
1400 BSTR$=0
M=1
RETURN

ERROR HANDLING SECTION FOLLOWS
6000 WRITE(6,6001) Func
6001 FORMAT(1000000 ERROR IN BARCODE PRODUCTION 000000 A4)
FM=P-1
WRITE(6,6010) SEQNUM,PM1,M(PM1),CHECK
6010 FORMAT(*,ROW='I3,' M('I4,' ') OPERAND='I3,' CHECKSUM='I4)
BSTR$=-1
RETURN
END
INTEGER FUNCTION COMP$(A$,LA,M,M1)

*** THIS IS THE MASTER INSTRUCTION INTERPRETATION ROUTINE FOR THE HP41C COMPI
*** HP41C COMPILER. THIS ROUTINE IS USED BY BOTH THE BAR CODE GENERATOR AND THE CALCU
*** INSTRUCTIONS ARE PASSED TO THIS ROUTINE ONE AT A TIME IN A TEXT STRING AS THE ARRAY M IS THE TOTAL ARRAY OF DECIMAL INTER
*** INTEGER KEY CODES (MACHINE INSTRUCTIONS), AND M1 IS THE POSITION WHERE THE NEXT DECODED MACHINE INSTRUCTION WILL BE PLACED. T
*** THE RETURN VALUE OF THE FUNCTION COMP$ IS SET AS FOLLOWS:
*** 1 = END STATEMENT FOUND, END COMPI
*** 0 = CONTINUE TO COMPIL
*** -1 = AN ERROR IN COMPILING THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)

COMMON/TEXT,IDIM,IPRT
COMMON/FLAGS/DONE,DECIMAL,DIGIT,ALPHA,INDIR,FLAG1,FLAG2 COMMON/CNTR/P3,P4,P5,P6,P7,P8,P9,S1,S2 COMMON/DONE,PO

LOGICAL DONE,PO

INTEGER P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 INTEGER A$[10]

INTEGER T1$(40),T2$(40),SAV$(40),WORK$(80)

INTEGER$ S5$(40),SS2$(40),SS3$(40),SS4$(40),SS5$(40)

INTEGER R1,R2,R3,R4,R5,R6,R7,R8,R9

INTEGER H(1),M

INTEGER*4 FUNCS,'COMP$'

INTEGER*2 ENDI$,'E','N','D$;

RCL$[3]','R$,'C$,'L$;

QUOTE$'/','/

APPEND$'/'

LBDI$[3]/'L$,'B$,'L$;

GTO[3]/'T$,'O$;

XEO[3]/'X$,'E$,'Q$;

XRO[3]/'X$,'R$,'Q$;

STO[3]/'S$,'T$,'D$;

* IND[3]/'I$,'N$,'D$;

COM00010

COM00020

COM00030

COM00040

COM00050

COM00060

COM00070

COM00080

COM00090

COM00100

COM00110

COM00120

COM00130

COM00140

COM00150

COM00160

COM00170

COM00180

COM00190

COM00200

COM00210

COM00220

COM00230

COM00240

COM00250

COM00260

COM00270

COM00280

COM00290

COM00300

COM00310

COM00320

COM00330

COM00340

COM00350

COM00360

COM00370

COM00380

COM00390

COM00400

COM00410

COM00420

COM00430

COM00440

COM00450

COM00460

COM00470

COM00480
C  2  END2(5)'E', 'N', 'D', './
C  2  AQUOTE(2)'A', './
C  2  MINUS(1)'/-
C  2  INTEGER(4) MRCL(5)/44, 32, 'R', 'C', 'L', 'MSTO(5)/145, 48, 'S', 'T', './
C  200 IF(IPRT, GE, 101) WRITE(6, 200) LA, (AS(I), I = 1, LA)
C  200 FORMAT('TRACE ', 'I3' ) COMPO540
C  200 IF(LA.GT.1) DIM.OR.LA.LT.0 GOTO 6000
C  
C  SET FLAGS AND INITIALIZE COUNTERS
C  INDIR = .FALSE.
C  
C  CHECK FOR NULL STRING ENTRY INTO COMP$
C  5  IF(LA .NE. 0) GOTO 10
C  5  COMP$ = 0 RETURN
C  
C  CHECK FOR CATEGORY THREE SPECIAL INSTRUCTIONS
C  10 IF(EQ$(AS, LA, RCL, 3, 3)) 6000, 15, 11
C  11  COMP$ = HEM$(AS, LA, M, M1, MRCL)
C  11  PDIGIT = .FALSE.
C  11  RETURN
C  15 IF(EQ$(AS, LA, STO, 3, 3)) 6000, 20, 16
C  16  MAKE A QUICK CHECK FOR THE STOP INSTRUCTION
C  16  IF(EQ$(AS(4), EQ, P)) GOTO 65
C  16  COMP$ = HEM$(AS, LA, M, M1, MSTO)
C  16  PDIGIT = .FALSE.
C  16  RETURN
C  20 IF(EQ$(AS, LA, LBL, 3, 3)) 6000, 25, 21
C  21  COMP$ = LBL$(AS, LA, M, M1)
C  21  PDIGIT = .FALSE.
C  21  RETURN
C  25 IF(EQ$(AS, LA, GTO, 3, 3)) 6000, 30, 26
C  26  COMP$ = GTO$(AS, LA, M, M1)
C  26  PDIGIT = .FALSE.
C  26  RETURN
C  30 IF(EQ$(AS, LA, XEQ, 3, 3)) 6000, 35, 31
31 \( \text{COMP} = \text{XEQ}(A, LA, M, M1) \)
\( \text{PDIGIT} = \text{.FALSE.} \)
RETURN

35 \( \text{IF(EQ}(A, LA, XRD, 3, 3)) \) 6000, 40, 36
\( \text{COMP} = \text{XRD}(A, LA, M, M1) \)
\( \text{PDIGIT} = \text{.FALSE.} \)
RETURN

40 \( \text{IF(EQ}(A, LA, END, 3, 3)) \) 6000, 45, 41
\( \text{COMP} = \text{END}(A, LA, M, M1) \)
\( \text{PDIGIT} = \text{.FALSE.} \)
RETURN

CHECK FOR ALPHABETIC ENTRY INSTRUCTION.

45 \( \text{IF}(A(1) \neq \text{QUOTE}) \text{ GOTO 50} \)
\( \text{COMP} = \text{ALP}1(A, LA, M, M1) \)
\( \text{PDIGIT} = \text{.FALSE.} \)
RETURN

CHECK FOR NUMERIC ENTRY INSTRUCTION.

50 \( \text{IF(NUM}(A, LA, TANSW) 6000, 55, 51 \)
\( \text{COMP} = \text{DIGT}(A, LA, M, M1) \)
\( \text{PDIGIT} = \text{.TRUE.} \)
RETURN

CHECK FOR CATEGORY ONE INSTRUCTION BY LOOKING FOR BLANK

55 \( \text{PL} = \text{POS}(A, LA, BLNK, 1, 1) \)
\( \text{IF}(\text{PL}) \) 6000, 65, 70

NC BLANK IN STRING IMPLIES HAVE FOUND ONE BYTE INSTRUCTION

65 \( \text{COMP} = \text{IONE}(A, LA, M, M1, 1) \)
\( \text{PDIGIT} = \text{.FALSE.} \)
RETURN
C

BLANK IN STRING MEANS MULTI-WORD INSTRUCTION, NOW EXTRACT PREFIX
70  IF(PARS$(A$,LA,SS1$,LSS1$)) 6000,65,75
C

CHECK FOR INDIRECT ADDRESSING
75  P6=POS$(A$,LA,IND,3,1)
1F(P6) 6000,80,76
76  INDIRET=TRUE.
1F(IPRI.GE.20)WRITE(6,235)
235  FORMAT(' DETECTED INDIRECT GOTO INSTRUCTION')
1F(LCUT$(A$,LA,3)) 6000,6080,80
C

COMPILE THE PREFIX OF A MULTI-WORD INSTRUCTION
30  COMP$=IONES(SS1$,LSS1,M,M1,2)
C

EXTRACT THE POSTFIX OF A MULTI-WORD INSTRUCTION
35  IF(PARS$(A$,LA,SS2$,LSS2$)) 6000,90,6090
C

COMPILE THE POSTFIX OF A MULTI-WORD INSTRUCTION
90  COMP$=MINO(COMP$,ITWO$(SS2$,LSS2,M,M1,INDIR))
PDIGIT=FALSE.
RETURN
C

ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
COMP$=-1
RETURN
6080 WRITE(6,6081)
6081 FORMAT(' ***** ERROR ***** ')
CCMP$=-1

INTEGER FUNCTION CON$(A$, LA$, $B$, $C$, LC)$

**STRING A$ AND STRING B$ ARE CONCATENATED AND PLACED IN C$.**

**IT IS FEASIBLE TO** CON$(A$, LA$, B$, A$, LA$)** OR**

**CON$(A$, LA$, B$, B$, LB)$**

**IN WHICH CASE THE APPROPRIATE STRING WILL BE REPLACED.**

**THE NUMBER OF CHARACTERS IN THE RESULTING STRING C$ IS RETURNED**

**AS THE VALUE OF THE FUNCTION CON$, UNLESS THERE IS A LOSS OF**

**CHARACTERS IN WHICH CASE THE NUMBER OF LOST CHARACTERS IS**

**RETURNED AS A NEGATIVE NUMBER.**

**IMPLICIT INTEGER(A-Z)**

**COMMON TEXT, IDIM, IPRT**

**INTEGER$2$ A$(IDIM), B$(IDIM), C$(IDIM)**

**INTEGER$4$ FUCN$*/CON***

**IF(IPRT.GE.10) WRITE(6,200)LA, (A$(I), I=1,LA)**

**IF(IPRT.GE.10) WRITE(6,201)LB, (B$(I), I=1,LB)**

**FORMAT('TRAC', 'I3', CON$: *', 110A1)**

**IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6030**

**IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000**

**LOSS=0**

**DETERMINE LENGTH OF RESULT**

ILC=LA+LB

**IF(ILC.LE.IDIM) GOTO 20**

**LOSS=ILC-IDIM**

**ILB=LB-LOSS**

ILC=IDIM

**IF(IPRT.GE.05) WRITE(6,202) LOSS**

**FORMAT('L', 'I3', 'CHARACTERS DURING CONCATENATION')**

202 **GOTO 25**

20 **ILB=LB**

25 **ILA=ILA**

**INDEX=ILC**

C C C

168
C    MCVE B$ INTO C$
C
   IF(ILB.LE.0) GOTO 40
   IND=ILB
   DC 35 I=1, ILB
   CS(I)$=B$(IND)
     IF(IPRT.GE.30) WRITE(6,207) IND,B$(IND),INDEX,C$(INDEX)
207    FORMAT(' MOVE B(*)I3,1)=','A1,' IS NOW C(*)I3,1)=','A1)
     IND=IND-1
     INDEX=INDEX-1
   CONTINUE

   MOVE A$ INTO C$
40
   IF(ILA.LE.0) GOTO 60
   IND=ILA
   DC 45 I=1, ILA
   CS(I)$=A$(IND)
   IF(IPRT.GE.30) WRITE(6,209) IND,A$(IND),INDEX,C$(INDEX)
209    FORMAT(' MOVE A(*)I3,1)=','A1,' IS NOW C(*)I3,1)=','A1)
     IND=IND-1
     INDEX=INDEX-1
   CONTINUE

   SET LENGTH OF C$ AND ASSIGN VALUE OF CONS$ AND RETURN.

60    LC=ILC
65   IF(IPRT.GE.20) WRITE(6,203) LC, (C$(I),I=1,LC)
203    FORMAT(' CONCAT: LC='*',I3,1)','=10A1)
   IF(LOSS.NE.0) GOTO 70
       CONS=1LC
600   GOTO 75
70    CONS=LOSS
75    RETURN

C    ERROR HANDLING SECTION FOLLOWS

6000  WRITE(6,6001) FNS
6001  FORMAT(' *** STRING LENGTH ERROR *** ','A4)
       CONS=-1
   RETURN

END
INTEGER FUNCTION DIGIT$(A$;LA;M;M1)

! THIS IS A FUNCTION THAT IS PART OF THE HP41C COMPILER. IT IS
! CALLED WHEN A DIGIT ENTRY INSTRUCTION IS ENCOUNTERED.
! THE RETURN VALUE OF THE FUNCTION DIGIT$ IS SET AS FOLLOWS:
! 0 = CONTINUE TO COMPILE
! -1 = AN ERROR IN COMPILING THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
COMMON/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
INTEGER A$[IDIM]
INTEGER PLUS/-/
INTEGER C$(13)/'0','1','2','3','4','5','6','7','8','9','1','2'/
2 INTEGER LC/13/
INTEGER FUNC$/'DIGT'/$
INTEGER M[1]
LOGICAL PDECIM,CHSFLG
IF(IPRT.GE.10) WRITE(6,200)IDM,(A$[1],I=1,LA)
200 FORMAT('TRACE',13,'DIGT$=',11(AI))
IF(ILA.GT.IDIM.OR.LA.LT.0) GOTO 6000
ADD A NULL INSTRUCTION BETWEEN ADJACENT DIGIT ENTRY INSTR.
IF(.NOT.PDIGIT) GOTO 400
ADJACENT DIGIT ENTRY INSTRUCTION FOUND
I/BYTE-1
M(M1)=BYTE
IF(IPRT.GE.20) WRITE(6,212)M1,BYTE
212 FORMAT('DIGT$=',15,'LENGTH OF THIS INSTR IS=',I3)
M1=M1+1
M(M1)=0
IF(IPRT.GE.20) WRITE(6,213)M1,M(M1)
213 FORMAT('DIGT$=',15,'NULL INSTR FOR PRECEEDING DIGIT ENTRY',
2 M1=M1+1
}
CHECK FOR DIGIT ENTRY INSTRUCTION PRECEDED BY PLUS SIGN.

400 IF(A$1).NE.PLUS)GOTO 450

NOTE THAT YOU GO AROUND THE FOLLOWING LINE IF THE FIRST
DIGIT IS NOT A PLUS SIGN OR IF THE PLUS SIGN IS ALL ALONE.
A PLUS SIGN BY ITSELF INDICATES ADDITION NOT A DIGIT
ENTRY INSTRUCTION. ADDITION IS COMPILED BY A TABLE LOOKUP.

CALL LCUT$(A$,LA,I)

SET THE LENGTH OF THE INSTRUCTION

450 IBYTE=LA
M(M1)=IBYTE
IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE

215 FORMAT(* DIGST$,15, ' LENGTH OF THIS INSTR IS',I3)
M1=M1+1

DO 35 I=1,LA
207 IZ=IND$(A$1,LA,C$,LC,LOC)
IF(IZ.NE.0)GOTO 20
WRITE(6,207)M1

20 FORMAT(* **** INVALID CHARACTER **** *5X,15)
DIGST$=-1
RETURN

20 M(M1)=IZ+15
208 IF(IPRT.GE.10)WRITE(6,208)M1,C$(IZ),M(M1)

2 FORMAT(* DIGST$,15, ' DIGIT ENTRY INSTR ',3X,A1,3X,
T75,13)
M1=M1+1

35 CONTINUE

DIGST$=0
RETURN

ERROR HANDLING SECTION FOLLOWS

6000 WRITE(6,6001) FNC$
6001 FORMAT(* *** STRING LENGTH ERROR *** ',A4)
6010 WRITE(6,6010) LA, LB, IDIM
       FORMAT(1, LA='I10', LB='I10', IDIM='I10)
       DIGIT$=-1
       RETURN
6999 WRITE(6,602)M1
602 FORMAT(1, '**** DIGIT ENTRY INSTR ERROR ***,5X,15)
       DIGIT$=-1
       RETURN
       END
INTEGER FUNCTION END$(A$, LA, M, M1)

**END00010
END00020
END00030
END00040
END00050
END00060
END00070
END00080
END00090
END00100
END00110
END00120
END00130
END00140
END00150
END00160
END00170
END00180
END00190
END00200
END00210
END00220
END00230
END00240
END00250
END00260
END00270
END00280
END00290
END00300
END00310
END00320
END00330
END00340
END00350
END00360
END00370
END00380
END00390
END00400
END00410
END00420
END00430
END00440
END00450
END00460
END00470

** END FUNCTION 
**

STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN END INSTRUCTION.

**END00080
**END00090
**END00100
**END00110
**END00120

THE RETURN VALUE OF THE FUNCTION END$ IS SET AS FOLLOWS:

0 = CONTINUE TO COMPILE
-1 = AN ERROR IN COMPILING THE INSTRUCTION.

**END00080
**END00090
**END00100
**END00110
**END00120

IMPLICIT INTEGER(A-Z)

COMMON/TEXT/IDIM, IPRT
COMMON/FLAGS/DONE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2

COMMON/CNTR/P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2

THIS SUBROUTINE Passes THE NUMBER OF ELEMENTS IN M VIA COMMON /CNTR.

LOGICAL DONE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2

INTEGER*4 P1, P2, P3, P4, P5, P6, P7, P8, S1, S2

INTEGER*2 A$(IDIM)

INTEGER*2 BLNK/ */

INTEGER*4 FUNC$/"END$"/

INTEGER*4 M(1)

IF(IPRT.GE.10) WRITE(6,200)LA$(A$(I), I=1, LA)

200 FORMAT(* TRAC *, I3, * ENDS *, * M1*, I10)(1)

IF(LA$.GT.IDIM.OR.LA$.LT.0) GOTO 6000

END INSTRUCTION IS THREE BYTES LONG. INDICATE LENGTH OF INSTR.

M(M1)=3

IF(IPRT.GE.20) WRITE(6,201)M1, IBYTE

201 FORMAT(* ENDS *, I5, * LENGTH OF NEXT INSTR IS *, I3)

M1=M1+1

LOAD PREFIX CODE FOR END INSTRUCTION. 193 IS USED INSTEAD OF 192.

END00450

WHILE 192 IS THE HP STANDARD OP-CODE FOR "END", THE 193 IS USED TO END00460

ENABLE 192 TO STAND FOR AN ALPHA LABEL INSTRUCTION. THIS USAGE END00470

IS STANDARD AMONG THE HP USERS GROUP PRACTICING SYNTHETIC PROGRAM-END00480
MING.

M(M1)=193
IF(IPRT.GE.10)WRITE(6,210)M1,M(M1)
210 FORMAT('END$','15',*END INSTR',T75,13)
M1=M1+1

PROVIDE TWO NULL INSTRUCTIONS TO RESERVE SPACE FOR THE LINK
pointers. ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
pointers which link them altogether into a label chain. this
chain is used to identify the position of labels and program
boundaries within the HP41CV memory. the chain of labels is
recompiled by the warm software, so the bytes containing the
chain are set to zero by this compiler.

M(M1)=0
IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211 FORMAT('END$','15',*TRAILING NULL INSTR',T75,13)
M1=M1+1
M(M1)=0
IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212 FORMAT('END$','15',*POINTER WILL BE RECOMPILED',T75,13)

NOTE NUMBER OF ELEMENTS IN THE MACHINE CODE ARRAY AND SET END FLAG
S2=M1
M1=M1+1
DONE=.TRUE.
WRITE(6,202)S2
202 FORMAT('ICOMPIILATION ENDED:','15',*MACHINE CODES GENERATED')
END$=1
RETURN

ERROR HANDLING SECTION Follows

6000 WRITE(6,6001) FUNC$
6001 FORMAT('*** STRING LENGTH ERROR *** ','A4')
WRITE(6,6010) LA, LB, IDIM
6010 FORMAT('LA=', '110', 'LB=', '110', ' IDIM=', '110')
END$=-1
RETURN
END
INTEGER FUNCTION EQS(A$,LA,B$,LB,NUM)

C** THIS FUNCTION TESTS FOR STRING EQUALITY. FOR THIS FUNCTION
C** THE RETURN VALUE IS CRUCIAL AS IT CONTAINS THE RESULTS OF THE
C** TEST FOR EQUALITY.
C** NUM DEFINES THE NUMBER OF CHARACTERS TO BE EXAMINED FOR
C** EQUALITY, STARTING FROM THE LEFT MOST POSITION OF BOTH
C** STRINGS. Thus, THE STRINGS:
C** A$='ABCDEF' AND B$='ABC'
C** WILL BE "EQUAL" IF TESTED WITH EQS(A$,LA,B$,LB,LB)
C** BUT "UNEQUAL" IF TESTED WITH EQS(A$,LA,B$,LB,LA)
C** TO TEST FOR ABSOLUTE EQUALITY, JUST ASSIGN NUM TO BE SOME
C** ARBITRARILY LARGE INTEGER, SAY 100. THE COMPARISON WILL
C** TERMINATE APPROPRIATELY AT THE END OF THE SHORTEST STRING.
C** IE. EQS(A$,LA,B$,LB,100M) WILL TEST ABSOLUTE EQUALITY
C** IT IS SUGGESTED THAT THIS ROUTINE BE USED IN AN ARITHMETIC
C** IF STATEMENT OF THE FORM:
C** IF EQS(A$,LA,B$,LB,LA)) 6002, 10, 20
C** WHERE: 6002 IS AN ERROR HANDLING ROUTINE
C** 10 IS THE ROUTINE WHEN STRINGS ARE NOT EQUAL
C** 20 IS THE ROUTINE WHEN STRINGS ARE EQUAL
C** COMMON/TEXT,IDIM,IPRT
C** INTEGER*2 A$(IDIM),B$(IDIM)
C** INTEGER*4 FUNC$=EQ$/
C** INTEGER*4 LA,LB,NUM
C** IF(IPRT.GE.10) WRITE(6,230)LA,NUM,(A$(I),I=1,LA)
200 FORMAT(* TRACE 'I',13,* EQ$(I',13,*)':',110A1)
201 FORMAT(' AND ',13,* ':',110A1)
199 FORMAT(' IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000'
1F(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C C
C LENGTH=NUM
IF((LENGTH.LE.LA).AND.(LENGTH.LE.LB))GOTO 10
IF(LA.NE.LB) GOTO 5
LENGTH=LA
GOTO 10

STRING CAN NOT BE EQUAL BECAUSE HAVE BEEN ASKED TO EXAMINE
MORE CHARACTERS THAN SMALLEST STRING IN A COMPARISON OF UNEQUAL
STRINGS.

EQ$=0
IF(IPRT.GE.20)WRITE(6,201)
RETURN

EXAMINE CHARACTERS ONE-BY-ONE TO TEST FOR EQUALITY.

DO 15 I=1,LENGTH
   IF(IPRT.GE.30) WRITE(6,201) I,A$(I),I,B$(I),
   FORMAT(1,COMPUTE A$('I',13.')='AL,' WITH B$(',13.',')='AL)
   IF(A$(I).EQ.B$(I)) GOTO 15
   EQ$=0
   IF(IPRT.GE.20)WRITE(6,202)
   FORMAT(' STRINGS FOUND UNEQUAL',13,' POSITION')
15 CONTINUE

IF YOU GET BELOW HERE THE STRINGS WERE FOUND TO BE EQUAL

EQ$=1
IF(IPRT.GE.20)WRITE(6,203)
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
WRITE(6,6010) LA,LB,IDIM
6010 FORMAT(' LA=',I10,'
   LB=',I10,'
   IDIM=',I10)
   EQ$=-9999
RETURN
END
INTEGER FUNCTION FIND$(AS, L, B, L, LOC)

**FIND AS IN B**

STRING B IS SEARCHED FOR THE FIRST OCCURRENCE OF A MATCH WITH
CHARACTER AS. AS IS NOT ALLOWED TO BE MORE THAN ONE CHARACTER
IN LENGTH.

SINCE B IS MOST LIKELY A TABLE OF CHARACTERS, IT IS ALLOWED,
AND MOST OFTEN IS OF A GREATER DIMENSION THAN IDIM, THE STANDARD
STRING DIMENSION.

THE VALUE OF THE FUNCTION FIND$ IS SET TO
- LOC (LOCATION OF FIRST MATCH IN B) IF MATCH FOUND
  0 NO MATCH IS FOUND
- 1 IF AN ERROR IS ENCOUNTERED.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 AS(I),BS(IDIM)
INTEGER*2 OBJECT
INTEGER*4 FUNC$,'FIND$'
INTEGER*4 LOC,FIND$

IF(IPRT.GE.10) WRITE(6,200)LA,AS(I)
200 FORMAT('TRACE',I3,' FIND$ : ',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000

OBJECT=AS(I)
INDEX=1
DC 25=1,LB

IF(IPRT.GE.40) WRITE(6,211) OBJECT,I,B$(I)
211 FORMAT('COMPARE OBJECT=',A1,' WITH B$(',I3,')=',A1)
IF(OBJECT.EQ.B$(INDEX)) GOTO 30
INDEX=INDEX+1
CONTINUE

NO MATCH FOUND
LOC=0
FIND$=0
IF(IPRT.GE.20) WRITE(6,201) LOC
INTEGER FUNCTION GTO$(A$; LA, M, H1)

**PARAMETERS**

- A$: Character string
- LA: Length of string
- M: Number of characters to strip
- H1: Instruction pointer

**DESCRIPTION**

- The function `GTO$` is used to identify a GTO instruction within a string.
- If the string contains a GTO instruction, the function returns the address of the instruction.
- If the string does not contain a GTO instruction, the function returns the address of the next instruction.

**RETURN VALUES**

- 0 = Continue to compile
- +1 = GTO instruction found
- -1 = Error in compiling

**IMPLEMENTATION**

- The function uses implicit integer variables and logical flags.
- It checks the string for the presence of GTO instructions and returns the appropriate value.

**Example Usage**

```plaintext
CALL GTO$(A$; LA, M, H1)
```

**Notes**

- The function assumes the absence of invalid characters in the string.
- It may require further adjustments for specific programming environments.
IF(A$(1).EQ.QUOTE) GOTO 80

PROCESS LOCAL LABELS, FIRST CHECK FOR INDIRECT GTO INSTRUCTION
PI=POS$(A$,LA,BLNK,1,1)
IF(PI) 6015,20,15

PROCESS GTO INDIRECT INSTRUCTION.
IF(EQ$(A$,LA,IND,3,3)) 6015,6020,16
CALL CUT$(A$,LA,PI)
IF(IPRT.GE.20)WRITE(6,235)
FORMAT('DETECTED INDIRECT GOTO INSTRUCTION');
INDIR=TRUE.
IBYTE=2
PREFIX=174

CHECK FOR NUMERIC OPERAND
IF(NUMC$(A$,LA,IANSW)) 6015,25,5

OPERAND MUST BE REGISTER X,Y,Z,T OR L OR A LOCAL ALPHA LABEL
DO 30 I=1,26
INDEX=I
IF(A$(I).EQ.LABEL(I)) GOTO 35
CONTINUE
Will fall through to this code if no valid label found
GOTO 6015

HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPERAND
INDEX=INDEX+101
GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

IF (IVAL$(A$, LA, INDEX) = 16015, 55, 55)

HAVE FOUND VALID NUMERIC LOCAL LABEL, CHECK FOR TWO BYTE INSTR

IF (INDEX .GT. 14 .OR. INDIR) GOTO 75

PROCESS A TWO BYTE INSTRUCTION, FIRST LOAD THE LENGTH OF INSTR

IBYTE = 2
M(M1) = IBYTE
IF (IPRT .GE. 20) WRITE(6, 213) M1, IBYTE
M1 = M1 + 1

HAVE FOUND VALID NUMERIC LOCAL LABEL < 15, LOAD "GTO" INSTRUCTION

M(M1) = 177 + INDEX
IF (IPRT .GE. 10) WRITE(6, 213) M1, M(M1)
FORMAT('GTO$, 15, 'TWO BYTE GTO INSTR', T75, 13)
M1 = M1 + 1

LOAD NULL INSTR FOR TWO BYTE GTO INSTR AND RETURN

M(M1) = 0
IF (IPRT .GE. 10) WRITE(6, 221) M1, M(M1)
M1 = M1 + 1
GTO$: = 0
RETURN

PROCESS THE GOTO INSTRUCTION (OPERAND > 14)
75  M(M1)=I_BYTE
10  IF(IPRT,GE,10)WRITE(6,210)M1,I_BYTE
210  FORMAT(' GTO$*,I5,* LENGTH OF NEXT INSTR IS',I3)
     M1=M1+1
C
C
C LOAD THE GOTO INSTR PREFIX
C
M(M1)=PREFIX
IF(IPRT,GE,10)WRITE(6,211)M1,M(M1)
211  FORMAT(' GTO$*,I5,* GOTO PREFIX INSTR',I75,I3)
     M1=M1+1
C
C
C LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTERS)
C
   IF(INDIR)GOTO 95
   M(M1)=0
   IF(IPRT,GE,10)WRITE(6,221)M1,M(M1)
221  FORMAT(' GTO$*,I5,* NULL FOR GOTO INSTR',I75,I3)
     M1=M1+1
C
C
C LOAD THE 2D OPERAND OF THE GOTO INSTR
C
   IF(INDIR)INDEX=INDEX+128
   M(M1)=INDEX
   IF(IPRT,GE,10)WRITE(6,212)M1,M(M1)
212  FORMAT(' GTO$*,I5,* GTO 2D OPERAND',I75,I3)
     M1=M1+1
     GTO4=0
     RETURN
C
C***********************************************************************
C***********************************************************************
C PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE
C
80  K=0
81  P2=POS$(A$+LA,QUOTE,1,2)
82  IF(P2) 6015,120,95
C
CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS
85  LEFT=LA-P2
     IF(LEFT) 5015,10),6015

DELETE THE ENDING QUOTE BY TRUNCATING THE STRING ONE CHAR
100  LA=LA-1

SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
   (LINE 120 ACCOUNTS FOR BEGINNING QUOTE STILL ON STRING)
120  LENGTH=LA-1 FOR GTO H=2 FOR LBL H=4 FOR XEQ H=2
     H=2
     IBYTE=H*LENGTH
     M(MI)=IBYTE
     IF(IPRT,GE,20)WRITE(6,210)M1,IBYTE
     M1=M1+1

HAVE FOUND VALID ALPHANUMERIC LABEL, LCAD "GTO" INSTRUCTION

PREFIX=29
C FOR GTO PREFIX=29 FOR LBL PREFIX=192 FOR XEQ PREFIX=30
   M(MI)=PREFIX
214  IF(IPRT,GE,10)WRITE(6,214)M1,M(MI)
     FORMAT("GTO",15,"ALPHA GTO INSTR",T75,I3)
     M1=M1+1

SET INDICATOR FOR NUMBER OF ALPHA CHARs IN LABEL
U=240
C FOR GTO U=240 FOR LBL U=241 FOR XEQ U=240
   M(MI)=U+LENGTH
   IF(IPRT,GE,10)WRITE(6,216)M1,M(MI)
216  FORMAT("GTO",15,"LENGTH CODE ALPH GTO",T75,I3)
     M1=M1+1
ADD ALPHABETIC CHARACTERS AND RETURN

I+O
LA=LENGTH+1
GTO$=ALPH$(A$,LA,M,M1)
RETURN

ERROR HANDLING SECTION FOLLOWS

6000  WRITE(6,6001) FUNC$
6001  FFORMAT(*** STRING LENGTH ERROR *** ,A4)
6010  WRITE(6,6010) LA,LB,IDIM
       FFORMAT( \r
LA='1,10,\r
LB='1,10,\r
IDIM='1,10)\r
       GTO$=-1\r
       RETURN
6015  WRITE(6,6016) \r
6016  FFORMAT(**** INVALID SECOND OPERAND IN GTO INSTR ****)\r
6020  WRITE(6,6021) \r
6021  FFORMAT(**** FOUND THREE OPERANDS, EXPECTING IND ****)
INTEGER FUNCTION IN$(AS,LAI)

*** ****************************

** STRING AS IS READ FROM UNIT IN.
** THE LENGTH OF AS IS AUTOMATICALLY COMPUTED, NOT COUNTING ANY
** LEADING OR TRAILING BLANKS, WHICH ARE TRIMMED AWAY.
** THE INPUT READER ASSUMES AN FIXED LENGTH INPUT RECORD OF 80
** CHAR.

** ****************************

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 AS(IDIM)
INTEGER*2 BLNK/''/
INTEGER*2 CARD(80)
INTEGER*4 FUNC/*IN*/
LOGICAL EOFILE(10)/10*.FALSE./
FORMAT(80A1)

C CHECK FOR END OF FILE
C IF(.NOT.EOFILE(IN)) GOTO 5
IN$=-1
LAI=0
IF(IPRT.GE.20) WRITE(6,201) IN
201 FORMAT('ATTEMPT TO READ AFTER END OF FILE ON UNIT ',12)
RETURN
C
C READ THE ACTUAL CARD
C READ (IN,100,END=999) (CARD(I),I=1,80)
IF(IPRT.GE.20) WRITE(6,222)(CARD(I),I=1,78)
222 FORMAT(' ',78A1)
C
C CHECK CARD FOR TRAILING BLANKS
C IM=0
OG 60 I=1,80
INDEX=81-I
IF(CARD(INDEX).NE.BLNK) GOTO 65
IM=IM+1
CONTINUE
IM=IM,EQ.0) GOTO 70
IF(IPRT,GE.20) WRITE(6,207) IM
207 FORMAT('FOUND',13,' TRAILING BLANKS IN INPUT STRING'
IF(IM,NE.80) GOTO 70
LA=0
IN$=0
IF(IPRT,GE.20) WRITE(6,208)
208 FORMAT('FOUND INPUT STRING IS ALL BLANKS'
IF(IPRT,GE.20) WRITE(6,200)LA,(A*(I),I=1,LA)
RETURN
70 IEND=80-IM

CHECK CARD FOR LEADING BLANKS
IM=0
DO 10 I=1,IEND
IF(CARD(I),NE.BLNK) GOTO 15
10 IM=IM+1
CONTINUE
IM=IM,EQ.0) GOTO 25
IF(IPRT,GE.20) WRITE(6,211) IM
211 FORMAT('FOUND',13,' LEADING BLANKS IN INPUT STRING'
IBEG=I+IM

DETERMINE LENGTH OF INPUT STRING
LA=IEND-IBEG+1
IF(LA.LE.IDIM) GOTO 30
25 IF(IPRT,GE.101) WRITE(6,216) LOSS
216 FORMAT('STRING TOO LONG FOR MAX STRING LENGTH. LOST',13)
LA=IDIM
IEND=IEND-LOSS

TRANSFER THE CARD CHARACTERS TO THE INPUT STRING.
INDEX=1
30 1=1BEG, IEND
DC 85 I=INDEX)CARD(I)
IF(IPRT.GE.30) WRITE(6,209) I,CARD(I),INDEX,A$(INDEX)
 209 FORMAT(' MOVE CARD('',13,'')='',A1,'' IS NOW C('',13,'')='',A1)
INDEX=INDEX+1
CONTINUE

CHECK FOR STRING ERROR AND RETURN

IF(IPRT.GE.20) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT(' TRACE *',13,* INS : ',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
INS=LA
RETURN

HANDLE END OF FILE CONDITION

ECFILE(IN)=.TRUE.
IN$=-1
LA=0
IF(IPRT.GE.20) WRITE(6,215)
215 FORMAT(' END OF FILE ENCOUNTERED')
RETURN

ERROR HANDLING SECTION follows

WRITE(6,6001) FUNC$
6000 FORMAT('*** ABNORMAL ENTRANCE *** ',A4)
6001 FORMAT('*** ABNORMAL ENTRANCE *** ',A4)
IN$=-1
RETURN
END
INTEGER FUNCTION IONE$(AS, LA, M, ML, IBYTE)

C* THIS FUNCTION IS THE TABLE DRIVEN INSTRUCTION LOOKUP. IT IS
C* USED TO TRANSLATE THE ONE BYTE INSTRUCTIONS IN THE HP41C
C* COMPILER.

C*

IMPLICIT INTEGER(A-Z)

COMMON/TEXT/IDIM, IPRT
COMMON/FLAGS/DONE, POPOINT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2
LOGICAL DONE, POPOINT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2
COMMON/TABLE/INST$, LINST, CODE, NINST
INTEGER#2 INST$(6, 111)
INTEGER#4 LINST$(111, CODE$(111), NINST
INTEGER#2 AS$(IDIM)
INTEGER#4 TNAME$"IONE$"
INTEGER#4 M$(1)
IF(IPRT$ .GE. 10) WRITE$(6, 200) LA, (AS$(I), I=1,LA)
FORMAT(* TRACE ' +', I3, ' IONE$ : +', I10, ' ) GOTO 6000

IF(LA$GT.IDIM.OR.LA$.LT.0) GOTO 6000

DO 50 I=1, NINST
LENGTH=INST$(I)
IF(LA$.NE.LENGTH) GOTO 50
DO 30 J=1, LENGTH
IF(AS$(J).NE.INST$(J, I11)) GOTO 50
CONTINUE

INSTRUCTION MATCHES SO CHECK TO SEE IF CORRECT NUMBER
OPERANDS. INSTRUCTIONS 143 OR LESS MUST HAVE ONLY ONE
OPERAND. INSTRUCTIONS 144 OR MORE MUST HAVE MORE THAN ONE.

IF( IbYTE$ .EQ. 1 .AND. CODE$(I) .LE. 143) GOTO 35
IF( IbYTE$ .EQ. 2 .AND. CODE$(I) .GT. 143) GOTO 35
GOTO 6020

LOAD CORRECTLY MATCHING VALUES TO MACHINE CODE ARRAY

M$(M) = IbYTE
IF(IPRT$ .GE. 20) WRITE$(6, 212) ML, IBYTE
212  FORMAT(  'IONES', 15,  ' LENGTH OF NEXT INSTR IS', 13)
     M1=M1+1
     M(M1)=CODE(1)
     IF(IPRT.GE.10) WRITE(6,210)M1,(INSTR(1:11),JJ=1,6),M(M1)
210  FORMAT( 'IONES', 15, ' INSTR', 75, 13)
     C       IF(LA.LT.LENGTH) GOTO 35
            LA=0
            IONES=0
            RETURN

THE FOLLOWING COMMENT LINES HAVE BEEN RETAINED TO FACILITATE USE OF
THE PROGRAM UNDER RULE THAT THE LENGTH OF A$ MAY BE MORE THAN THE
LENGTH OF THE MATCH STRING. CODE HAS BEEN TESTED AND PROVEN TO
SELECT FIRST SUBSTRING MATCH IN TABLE.

C35  LEFT=LA-LENGTH
     LSTART=LENGTH+1
     IF(SEG(A$,LA$,A$,LSTART,LEFT),GT.0) GOTO 40
     IONES=0
     RETURN
C40  IONES=LA
     RETURN

INSTRUCTION DOES NOT MATCH SO CHECK NEXT INSTRUCTION
C50  NO MATCH FOUND

C60  WRITE(6,215)
     FORMAT( **** UNRECOGNIZABLE INSTRUCTION ****)
     M(M1)=IBYTE
     IF(IPRT.GE.20) WRITE(6,212)M1,IBYTE
     M1=M1+1
     M(M1)=0
     IF(IPRT.GE.10) WRITE(6,211)M1,M(M1)
211  FORMAT(  'IONES', 15,  ' NULL INSTR', 75, 13)
     C       IONES=-1
            RETURN
C
C  ERROR HANDLING SECTION Follows
C
6000 WRITE (6, 6001) FUNC$  ION00970
6001 FORMAT( '*** STRING LENGTH ERROR ***', A4)  ION00980
6010 FORMAT( 'LA=', 'I10 ',', LB=', 'I10 ',', IDIM=', 'I10)  ION00990
  ICN$=-1  ION01000
  RETURN  ION01010
6020 WRITE (6, 6021)  ION01020
6021 FORMAT( '*** LENGTH OF INSTRUCTION DOES NOT MATCH NUMBER OPERNDS')  ION01030
  ICN$=-1  ION01040
  RETURN  ION01050
END  ION01060
ION01070
ION01080
ION01090
INTEGER FUNCTION IRDR$ (I$, VAL)

C** THIS FUNCTION READS A FREE FORMAT VALUE AND CONVERTS IT TO
** AN INTEGER.
C**
C**
C**
C**
C**
C**
C**
C**
C**
C**
C**

C*****************************************************************************
C IMPLICIT INTEGER(A-Z)
C COMMON/TEXT/IDIN, IPRT
C INTEGER*2 A$ (80)
C INTEGER*2 BLNK/' ', ZERO/' 0', / MINUS/' -'/
C DATA LL/256/
C INTEGER*4 RVAL, SIGN, IFN, FRAC
C INTEGER*4 IRDR$, VAL, I$
C IF(IPRT, GE, 20) WRITE(6, 200) I$
C 200 FORMAT(* TRACE IRDR$ ', 15)
C
C
C IF(IN$(A$, LA, I$))6000, 12, 12
C
C 12 SIGN=1
C IFN=0
C 15 DI II=1, LA
C IF(IPRT, GE, 30) WRITE(6, 213) A$(II)
C 213 FORMAT(* EXAMINING ', A', FOR INTEGER VALUE*)
C IF(A$(II).EQ. MINUS) GO TO 10
C IF(A$(II).EQ. BLNK) GOTO 1
C 20 ITEMP=A$(II)-ZERO
C ITEMP=ITEMP/LL
C IF((ITEMP.GT.9).OR. (ITEMP.LT.0)) GOTO 6007
C IF(IPRT, GE, 30) WRITE(6, 216) ITEMP
C 215 FORMAT(* FOUND NUMERIC DIGIT*, 12)
C IFN=IFN+10+ITEMP
C GO TO 1
C 10 SIGN=1
C IF(IPRT, GE, 30) WRITE(6, 214)
C 214 FORMAT(* FOUND MINUS SIGN*)
C CONTINUE
C 217 FORMAT(* SUBROUTINE IRDR$ RETURNING VALUE ', 120)
C IF(IPRT, GE, 30) WRITE(6, 2225) IFN, SIGN
C 2225 FORMAT(* SET FINAL SIGN OF ', 120, * WITH ', 120)
C 660 IRDR$=SIGN*IFN
VAL=IRD$R$  
IF(IPRT.GE.20)WRITE(6,217) VAL  
RETURN  

C C C  
ERROR HANDLING SECTION FOLLOWS  

6000 WRITE(6,6001)  
6001 FORMAT(' *** END OF FILE DETECTED *** ',A4)  
IRD$=-1  
STOP  

6007 WRITE(6,6008)(A$(I),I=1,LA)  
6008 FORMAT(' *** ATTEMPT TO FIND INTG VALUE OF ALPHABETIC STRING:*** 
STOP  
1  
END
INTEGER FUNCTION ITWO$(A$,LA, M, M1, INDIR)

STRING A$ IS A POSTFIX FOR A MULTI-WORD INSTRUCTION.

THIS ROUTINE WILL EXAMINE THE POSTFIX AND RETURN A DECIMAL

VALUE INTERPRETATION OF THE POSTFIX.

INDIRECT INSTRUCTIONS WILL HAVE THE POSTFIX APPROPRIATELY

SET WITH THE HIGH ORDER BIT ON, AS REQUIRED BY THE INDIR FLAG

THE RETURN VALUE OF THE FUNCTION ITWO$ IS SET AS FOLLOWS:

0 = CONTINUE TO COMPILE

-1 = AN ERROR IN COMPILING THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPIR
INTEGER*2 A$(IDIM)/
INTEGER*2 BLNK$/
'T$', 'U$', 'V$', 'W$', 'X$', 'Y$', 'Z$)/
INTEGER*4 FUNC$'$ITWO$'/
INTEGER*4 M(1)
LOGICAL INDIR
IF(ING$.GE.10) WRITE(6,200)LA,(A$(I),I=) LA)

200 FORMAT('TRACE ',I3,' ITWO$: ',110A$)
IF(LA.GT.IDIM.0R.LA.LT.0) GOTO 6000

CHECK FOR BLANK INDICATING A BUGUS THIRD OPERAND

IF(TRIM$(A$,LA)) 6015, 6030, 10
IF(POST$(A$,LA,BLNK,1,1)) 6015, 20, 6020

CHECK FOR NUMERIC OPERAND

IF(NUMC$(A$,LA, IANSW)) 6015, 25, 50
OPERAND MUST BE REGISTER X, Y, Z, T OR L OR A LOCAL ALPHA LABEL

DO 30 I=1,26
  INDEX=I
  IF(A$(I).EQ.LABEL(I)) GOTO 35
CONTINUE

WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND
GOTO 6015

HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-
AND THEN GOTO PROCESS SECTION TO ACTUALLY LOAD BYTE.

INDEX=INDEX+101
GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

IF(I VAL$(AS,LA,INDEX))6015,75,75

HAVE FOUND VALID POSTFIX, CHECK FOR INDIRECT INSTRUCTION

IF(INDIR) INDEX=INDEX+128

LOAD THE SECOND OPERAND IN THE MACHINE CODE ARRAY

M(MI)=INDEX
IF(IPRT.GE.10) WRITE(6,212) M1,M(MI)
212 FORMAT(' ITWO$',15,') 2D OPERAND ','T75,13)

M1=M1+1

CLEAN-UP AND RETURN

INDIR=.FALSE.
ITWO$=0
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
6001  FORMAT( '*** STRING LENGTH ERROR *** ',A4)
WRITE(6,6010) LA, LB, IDIM
6010  FORMAT( 'LA=',I10,',  LB=',I10,',  IDIM=',I10)
ITWO$=-1
RETURN
WRITE(6,6016)
6016  FORMAT( '***** INVALID SECOND OPERAND IN INSTR *****')
ITWO$=-1
RETURN
WRITE(6,6021)
6021  FORMAT( '***** FOUND THREE OPERANDS, EXPECTING IND *****')
ITWO$=-1
RETURN
WRITE(6,6031)
6031  FORMAT( '***** FOUND SECOND OPERAND BLANK *****')
ITWO$=-1
RETURN
END
**INTEGER FUNCTION IVAL$(A$, LA$, VAL$)**

**CONVERTS A NUMERIC TEXT STRING TO INTEGER NUMERIC VALUE.**

**IMPLICIT INTEGER(A-Z)**

**COMMON/TEXT/LIDIM, IPRT**

**INTEGER*2 A$[I][DIM$]**

**INTEGER*2 BLNK$,','/ZERO,'/'MINUS,'-'/'**

**INTEGER*4 FNC$,'/'VAL$**

**DATA LL$'/256/'**

**INTEGER*4 RVAL, SIGN, IFN, FRAC**

**IF(IPRT.GE.10) WRITE(6,200) LA$, (A$[I]), I=1, LA$**

200 **FORMAT(' TRACE 'I3', 'IVAL$ 'I10.1')**

**IF(LA$.GT.LIDIM.OR.LA$.LT.0) GOTO 6000**

**C**

**SIGN=1**

**IFN=0**

15 **DC [I=1,LA$**

213 **IF(PRT.GE.30) WRITE(6,213)A$[II]**

216 **FORMAT(' EXAMINING 'A[, FOR INTEGER VALUE')**

**IF(A$[II].EQ.MINUS) GO TO 10**

**IF(A$[II].EQ.BLNK) GOTO 1**

20 **ITEMP=A$[II].ZERO**

**ITEMP=ITEMP/LL**

**IF(ITEMP.GT.9. OR. (ITEMP.LT.0)) GOTO 6007**

214 **IF(FOUND NUMERIC DIGIT, I2) IFN=IFN+1$ITEMP**

**IF/IPRT.GE.30) WRITE(6,214)**

**FORMAT(' FOUND MINUS SIGN')**

**CONTINUE**

217 **IF(SUBROUTINE IVAL$ RETURNING VALUE 'I20) IF(PRT.GE.30) WRITE(6,2225) IFN$SIGN**

**IF(SUBROUTINE IVAL$ RETURNING VALUE 'I20) IFN$SIGN**

2225 **FORMAT(' SET FINAL SIGN OF ', I20, ' WITH', I20)**
660 IF VAL$=SIGN$ THEN RETURN
   IF VAL$=1 THEN WRITE(6,1271) VAL$
   FORMATS$=1
   STOP
   FORMATS$=1
   STOP
   END
INTEGER FUNCTION LBL$(A$,LA,M,M1)

STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN LBL INSTRUCTION.
THE RETURN VALUE OF THE FUNCTION LBL$ IS SET AS FOLLOWS:

0 = CONTINUE TO COMPILE
-1 = AN ERROR IN COMPILING THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
COMMON/FLAGS/DONE,P,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
LOGICAL DONE,P,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
INTERGER* P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTERGER* A$(IDIM)
INTERGER* S$(201)
INTERGER* LESS
INTERGER* BLNK/"
INTERGER* QUOTE/"
INTERGER* $LABEL(26)'/A'/'B'/'C'/'D'/'E'/'F'/'G'/'H'/'I'/'J';
INTERGER* FUNC'/LBL'/
INTERGER* M(1)
IF(IPRT,GE,10) WRITE(6,200)LA,(A$I,I=1,LA)
200 FORMAT('TRACE ...',I3,I,LBL$ T:110A1)
IF(LA,GT,IDIM.OR.LA,LT,0) GOTO 6000

CALL LCUT$(A$,LA,3)
IF(TRIM$(A$,LA)) 6015,6015,10

CHECK FOR ALPHANUMERIC VERSUS LOCAL LABELS

IF(A$(1),EQ,QUOTE) GOTO 80

PROCEED LOCAL LABELS, FIRST CHECK FOR NUMERIC OR SINGLE CHAR ALPHABET.
20 IF(LA-2) 25,50,6015

LENGTH OPERAND IMPLIES SINGLE CHARACTER ALPHA LOCAL LABEL

25 DO 30 I=1,26
   INDEX=I
   IF(A$=I).EQ.LABEL(I)) GOTO 35
   CONTINUE

30 WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND
   GOTO 6015

HAVE FOUND A MATCH IN LOCAL LABEL TABLE; SET VALUE OF SECOND OPER-
AND. THEN GOTO PROCESS A TWO BYTE INSTRUCTION.

35 INDEX=INDEX+101

PROCESS A TWO BYTE INSTRUCTION

40 I=BYTE

41 M(I)=I BYTE

42 IF(IPRT.GE.20) WRITE(6,210) M(I),I BYTE

210 FORMAT(*,BLK$,15,* LENGTH OF NEXT INSTR IS$,I3)

43 M(I)=M(I)+1

LOAD TWO BYTE LBL INSTR (EITHER SINGLE CHAR ALPHA OR 2 DIGIT NUM)

44 M(I)=207

211 IF(IPRT.GE.10) WRITE(6,211) M(I),M(I)

211 FORMAT(*,BLK$,15,* TWO BYTE LBL INSTR$,T75,I3)

45 M(I)=M(I)+1

46 M(I)=INDEX

212 IF(IPRT.GE.10) WRITE(6,212) M(I),M(I)

212 FORMAT(*,BLK$,15,* LBL 2D OPERAND $,T75,I3)

47 M(I)=M(I)+1

48 LBL$=0

RETURN
LENGTH OF OPERAND IMPLIES MUST BE A NUMERIC LOCAL LABEL

50 IF(IVAL$(A$,LA,INDEX))6015,55,55

55 HAVE FOUND VALID NUMERIC LOCAL LABEL, CHECK FOR ONE BYTE INSTR
   IF(INDEX.GT.14) GOTO 40

C PROCESS A ONE BYTE INSTRUCTION; FIRST LOAD THE LENGTH OF INSTR
   IBYTE=1
   M[M1]=IBYTE
   IF(IPRINT.GE.20)WRITE(6,210)M1,IBYTE
   M1=M1+1

C HAVE FOUND VALID NUMERIC LOCAL LABEL <15, LOAD "LBL" INSTRUCTION
   M[M1]=1+INDEX
   IF(IPRINT.GE.10)WRITE(6,213)M1,M(M1)
   FORMAT('LBL$,I15,'"ONE BYTE LBL INSTR",T75,13)
   M1=M1+1
   LBL$=0
   RETURN

C PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE
   K=0
   P2=POS$(A$,LA,QUOTE,1,2)
   IF(P2) 6015,120,85

C LOOK FOR ANOTHER QUOTE
   P4=POS$(A$,LA,QUOTE,1,PL)
   IF(P4) 6015,95,90
FCUND ANOTHER (THIRD OR MORE) QUOTE
90    P2=P4
      GOTO 85

CHECK AFTER LAST QUOTE FOR KEY ASSIGNMENT CODE
95    LEFT=LA-P2
      IF (LEFT) 6015,100,105
C100    LA=LA-1
      GOTO 120
C105    RSTART=P2+1
      CALL SEG$(A$,LA,SSL$,LSS$,PSTART,LEFT)
      K=IVAL$(SSL$,LSS$,NO)
      LA=LA-LEFT-1

SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
120    LENGTH=LA-1
      IBYTE=4+LENGTH
      M(M1)=IBYTE
      IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
      M1=M1+1

HAVE FOUND VALID ALPHANUMERIC LABEL, LOAD "LBL" INSTRUCTION
C192    M(M1)=192
      IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
214    FORMAT(1,LBL$",15,"ALPHA LBL INSTR",175,13)
      M1=M1+1

PROVIDE ONE NULL INSTRUCTION TO RESERVE SPACE FOR THE LINK
PCINTER. ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
POINTERS WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN. THIS
CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
BOUNDARIES WITHIN THE HP41CV MEMORY. THE CHAIN OF LABELS IS
RECOMPiled BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE
CHAIN ARE SET TO ZERO BY THIS COMPILER.

M(M1) = 0

IF(IPRT.GE.10) WRITE(6,215) M1,M(M1)

IF(IPRT.GE.10) WRITE(6,216) M1,M(M1)

WRITE(6,217) M1,M(M1)

WRITE(6,218) M1,M(M1)

IF(K.GE.0) GOTO 130

IF(K.GT.0) GOTO 135

K1 = 8

K = INT(K)

A1 = K/10

B1 = MOD(K,10)

K = 16*B1 + A1

M(M1) = K

IF(IPRT.GE.10) WRITE(6,219) M1,M(M1)

WRITE(6,220) M1,M(M1)

WRITE(6,221) M1,M(M1)

WRITE(6,222) M1,M(M1)

WRITE(6,223) M1,M(M1)

WRITE(6,224) M1,M(M1)

WRITE(6,225) M1,M(M1)

WRITE(6,226) M1,M(M1)

WRITE(6,227) M1,M(M1)

WRITE(6,228) M1,M(M1)

WRITE(6,229) M1,M(M1)

WRITE(6,230) M1,M(M1)

WRITE(6,231) M1,M(M1)

WRITE(6,232) M1,M(M1)

WRITE(6,233) M1,M(M1)

WRITE(6,234) M1,M(M1)

WRITE(6,235) M1,M(M1)

WRITE(6,236) M1,M(M1)

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RE
C
C ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$  
6001 FORMAT(16,6010) STRING LENGTH ERROR *** ',A4)  
6010 FORMAT(16,6010) LA, LB, IDIM  
               LA=' ',II0, '  LB=' ',II0, '  IDIM=' ',II0)  
               -1  
               RETURN  
6015 WRITE(6,6016)  
6016 FORMAT(16,6010) INVALID SECOND OPERAND IN LBL INSTR *****)  
               -1  
               RETURN  
               END
INTEGER FUNCTION MEM$(A$,LA,M,M1,WHICH)

** C** ** STRING A$ CONTAINS AN MEMORY INSTRUCTION, EITHER AN STO OR A ** RCL **
** C** ** THE RETURN VALUE OF THE FUNCTION MEM$ IS SET AS FOLLOWS: **
** C** ** 0 = CONTINUE TO COMPILE **
** C** ** -1 = AN ERROR IN COMPILING THE INSTRUCTION. **

** C** ** IMPLICIT INTEGER(A-Z) **
** C** ** COMMON/TEXT/IDIM, IPRT **
** C** ** COMMON/FLAGS/DONE,P,DIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2 **
** C** ** COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 **
** C** ** LOGICAL DONE,P,DIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2 **
** C** ** INTEGER P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 **
** C** ** INTEGER A$(IDIM) **
** C** ** INTEGER BLNK/* */ **
** C** ** INTEGER IND(3)* */ 'N'*, 'O'*/ **
** C** ** INTEGER C $(5)* */ 'U', 'V', 'X', 'L' */ **
** C** ** INTEGER LC/,5/ **
** C** ** INTEGER WHICH(5) **
** C** ** INTEGER FUNCS*/MEM$ */ **
** C** ** INTEGER M(1) **
** C** ** IF(IPRT.GE.10) WRITE(6,200)LA,(A$(1),1=1,LA) **
** C** ** 200 FORMAT(* TRACE *,I3,* MEM$ : *,110A1) **
** C** ** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000 **

** C** ** STRIP STRING OF "MEM" CHARACTERS. **
** C** ** IF(LCUT$(A$,LA,3)) 6015,6015,7 **
** C** ** IF(TRIM$(A$,LA)) 6015,6015,10 **

** C** ** ESTABLISH MOST LIKELY INSTR LENGTH AND PREFIX **
** C** ** IBYTE=2 **
** C** ** PREFIX=WHICH(1) **
** C** ** H=WHICH(2) **

** C** ** CHECK FOR INDIRECT ADDRESS **
MOVE PTR=(A$) TO LA

IF(P$) 6000,40,21
   INDIR=TRUE
   IF(LCUT$(A$,LA,3)) 6000,6015,22
   IF(TRIM$(A$,LA)) 6015,6015,40

CHECK FOR NUMERIC SECOND OPERAND
   IF(NUMC$(A$,LA,IANSW)) 6015,45,60

PROCESS NON-NUMERIC SECOND OPERAND
   IF(LA.GT.1) GOTO 6015
   IF(LZ=0) GOTO 46
   WRITE(6,207)A$(1)
   FORMAT('******** INVALID CHARACTER ********',5X,A1)
   MEM$=-1
   RETURN
   POSTFX=LZ+111
   GOTO 100

PROCESS NUMERIC POSTFIX OPERANDS
   POSTFX=IVAL$(A$,LA,VAL)

CHECK FOR ONE BYTE VERSUS TWO BYTE NUMERIC OPERANDS
   IF((POSTFX.GT.15).CR.(INDIR)) GOTO 75

PROCESS ONE BYTE NUMERIC SECOND OPERANDS
   IF(BYTE=1)
      PREFIX=POSTFX+H
   GOTO 100

GOTO 100
PROCESS TWO BYTE NUMERIC OPERANDS

SET THE LENGTH OF THE INSTRUCTION

IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215 FORMAT('"MEM$' ,15,' LENGTH OF THIS INSTR IS','I3)
M1=M1+1

ENCODE THE PREFIX OF THIS INSTRUCTION

IF(IPRT.GE.10)WRITE(6,211)M1,(WHICH(I),I=3,5),M(M1)
211 FORMAT('"MEM$' ,15,'',3A1,' INSTR',T75,I3)
M1=M1+1

ENCODE THE POSTFIX OF THIS INSTRUCTION

IF(IPRT.GE.10)WRITE(6,221)M1,(WHICH(I),I=3,5),M(M1)
221 FORMAT('"MEM$' ,15,'',3A1,' INSTR POSTFIX',T75,I3)
M1=M1+1

MEM$=0
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
6001 FORMAT('*** STRING LENGTH ERROR *** ','A4)
WRITE(6,6010) LA, LB, DIM
6010 FORMAT('LA=',110,' LB=',110,' DIM=',110)
MEM$=-1
RETURN

WRITE(6,6016)
WRITE(6,6016)
WRITE(6,6016)
WRITE(6,6016)
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN

MEM$=-1
RETURN
INTEGER FUNCTION NEWPG$(LINCNT, NUMPAGE, TITLE$, LTITLE, MTITLE)

C***********************************************************************
C** THIS SUBROUTINE IS USED TO PLACE MAKE THE OUTPUT LISTING
C** AS ATTRACTIVE AS POSSIBLE. IT PERFORMS ALL THE HOUSEKEEPING
C** FUNCTIONS ASSOCIATED WITH GOING TO A NEW PAGE.
C***********************************************************************
C** IT SHOULD BE CALLED BY THE USE OF A STATEMENT SUCH AS
C**
C** IF(MOD(LINCN'T, PAGENUMBER), EQ, 0)
C** 1 CALL NEWPG$(LINCNT, PAGENUMBER, TITLE$, LTITLE, MTITLE)
C***********************************************************************

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 TITLE$(LTITLE, MTITLE)
INTEGER*2 BLNK/' '/
INTEGER*2 CARD(80)
INTEGER*4 FUNC$/'NEWPG$ /
297 FORMAT(//, ' ')
299 FORMAT('1111',/116A1)
200 FORMAT(' ',132A1)

PRINT THE OUTPUT PAGE HEADING
WRITE(6,299)
IF(MTITLE.LE.0) GOTO 75
WRITE(6,201)(TITLE$(JJ, JJ), JJ=1, LTITLE)
IF(MTITLE.LE.0) GOTO 75
DC 50 II = 2*MTITLE
WRITE(6,201)(TITLE$(JJ, II), JJ=1, LTITLE)
CONTINUE

UPDATE THE PAGE COUNTER AND RESET THE LINE COUNTER
NUMPAGE=NUMPAGE+1
LINCNT=MTITLE
WRITE(6,297)
EXIT

NEWPG$=NUMPGE
RETURN

ERROR HANDLING SECTION Follows

6000 WRITE(6,6001) FUNC$
6001 FORMAT(*** PAGE OUTPUT ERROR *** ',',A4)
NEWPOS$=-1
RETURN
END
INTEGER FUNCTION NUMCS$(A$1, IANSW)
C******************************************************************************
C** IF (LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C******************************************************************************
C** IMPLICIT INTEGER(A-Z)
C** COMMON/TEXT/IDIM, IPRT
C** INTEGER*2 A$(IDIM)
C** INTEGER*4 FUNC$/'NUMCS'/
C** INTEGER*2 DIGIT(15)/'0','1','2','3','4','5','6','7','8','9,'
C** IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I))=I,LA)
C** FORMAT('TRACE ',I3,' NUMCS$: ',110A1)
C** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C******************************************************************************
C DC 10 I=1,LA
C IF(LA.GE.1) GOTO 3
C NOT ALLOWED TO EXAMINE LAST 5 CHARS IF LA IS ONE.
C LOOK=10
C GOTO 4
C LOOK=15
C DO 5 J=1,LOOK
C IF(A$(J).EQ.DIGIT(J)) GOTO 10
C CONTINUE
C HAVE FOUND A NON-NUMERIC CHARACTER
C IANSW=0
C NUMCS=0
C IF(IPRT.GE.20) WRITE(6,201)
C FORMAT(' STRING DETERMINED TO BE NON-NUMERIC ')
C RETURN
C CHARACTER FOUND TO BE NUMERIC, TAKE NEXT CHARACTER
C CONTINUE
C IF YOU GET TO HERE THE STRING MUST BE ALL NUMERIC.
C******************************************************************************
IANSW=1
NUMCS=1
IF(IPRT,GE,20) WRITE(6,202)
202 FORMAT(' STRING FOUND TO BE NUMERIC')
RETURN

ERROR HANDLING SECTION FollowS

WRITE(6,6001) FUNC$  
6001 FORMAT('*** STRING LENGTH ERROR *** ',A4)
WRITE(6,6010) LA, LB, IDIM
6010 FORMAT('LA=',' I10,2', 'LB=',' I10,2', 'IDIM=',' I10,2')
NUMCS=-1
RETURN
END
INTEGER FUNCTION PARS$(A$, $LA$, $B$, $LB$)

**STRING A$ IS SEARCHED FOR THE OCCURRENCE OF THE 1ST NON-LEADING BLANK. THEN A$ IS SPLIT INTO TWO SUBSTRINGS, THE LEADING TOKEN (FIRST WORD) IS PLACED IN B$ AND THE REMAINDER IS PLACED IN A$.

**THE NUMBER OF NON-BLANK CHARACTERS REMAINING IN A$ AFTER THE REMOVAL OF THE FIRST WORD IS RETURNED AS THE FUNCTION VALUE.

**AN ATTEMPT TO PARSE A NULL STRING WILL RESULT IN A SPECIAL CHARACTER BEING PLACED IN THE 1ST POSITION OF A$. A SUBSEQUENT PARSE FUNCTION ON A NULL STRING WILL RESULT IN A FATAL ERROR.

**THIS FEATURE IS INTENDED TO PREVENT UNCONTROLLED LOOPS OF THE PARSE FUNCTION. THE SPECIAL CHARACTER USED IS KNOWN ONLY TO THIS ROUTINE AND MAY BE USED AS A REGULAR CHARACTER FOR IN ANY STRING OF LENGTH GREATER THAN ZERO.

**IMPLICIT INTEGER(A-Z)

COMMON/TEXT/IDIM, IPRT, INTEGER*2 A$, IDIM, B$, IDIM

INTEGER*2 BLNK/1 /

INTEGER*2 HALT/0 /

INTEGER*2 FUNC$/'/

IF (IPRT.GE.10) WRITE(6, 200) LA, (A$(I) $I = 1, LA)

200 FORMAT('TRACE ', I3, '$ ', A$; $ I = 1, I)

IF (LA.GT.IDIM.OR.LA.LT.0) GTO 6000

CHECK TO SEE IF INPUT STRING IS NULL

IF (LA.NE.0) GOTO 5

IF (IPRT.GE.20) WRITE(6, 204)

204 FORMAT('ATTEMPTED TO PARSE A NULL STRING.')

IF (A$(I).EQ.HALT) GOTO 6005

A$(I) = HALT

PARS$ = 0

LB = 0

RETURN

TRIM THE INPUT STRING OF LEADING BLANKS
5 IM=0
   DO 10 I=1,LA
   IF(A$[I].NE.BLNK) GOTO 15
   IM=IM+1
10 CONTINUE
15 IF(IM.EQ.0) GOTO 25
   IF(IPRT.GE.20) WRITE(6,201) IM
201 FORMAT('FOUND',I3,' LEADING BLANKS IN INPUT STRING')
   IF(IM.LT.LA) GOTO 20
   LA=0
   LB=0
   PARS$=0
   IF(IPRT.GE.20) WRITE(6,202)
202 FORMAT('FOUND INPUT STRING IS ALL BLANKS')
   RETURN
20 LEFT=LA-IN
   DO 23 I=1,LEFT
   A$(I)=A$(I+IM)
23 CONTINUE
   LA=LEFT

LOCATE THE FIRST NON-LEADING BLANK IN A$ (THEREBY DETERMINE LB)
25 LB=0
   DO 30 I=1,LA
   IF(A$[I].EQ.BLNK) GOTO 35
   LB=LB+1
30 CONTINUE

CONSTRUCT TOKEN
35 DO 45 I=1,LB
   B$(I)=A$(I)
45 CONTINUE
   IF(IPRT.GE.20) WRITE(6,205) LB,(B$(I),I=1,LB)
205 FORMAT('PARS$ FOUND TOKEN',I3,' **',60A1)

REMOVE TOKEN FROM FRONT OF INPUT STRING

LEFT=LA-LB-1
   IF(LEFT.GT.0) GOTO 50
   LA=0
   GOTO 75
50 DO 55 I=1,LEFT

PAR00490
PAR00500
PAR00510
PAR00520
PAR00530
PAR00540
PAR00550
PAR00560
PAR00570
PAR00580
PAR00590
PAR00600
PAR00610
PAR00620
PAR00630
PAR00640
PAR00650
PAR00660
PAR00670
PAR00680
PAR00690
PAR00700
PAR00710
PAR00720
PAR00730
PAR00740
PAR00750
PAR00760
PAR00770
PAR00780
PAR00790
PAR00800
PAR00810
PAR00820
PAR00830
PAR00840
PAR00850
PAR00860
PAR00870
PAR00880
PAR00890
PAR00900
PAR00910
PAR00920
PAR00930
PAR00940
PAR00950
PAR00960
A$(I) = A$(I+LB+1)  
CONTINUE  
LA = LEFT  
CHECK $A FOR TRAILING BLANKS  
IF(LA.EQ.0) GOTO 75  
I$ = 0  
DO 60 I = 1, LA  
INDEX = LA - I + 1  
IF(A$(INDEX) .NE. BLNK) GOTO 65  
IM = IM + 1  
CONTINUE  
IF(IM.EQ.0) GOTO 75  
IF(IPRT .GE. 20) WRITE(6, 207) IM  
FORMAT(' FOUND ', IM, ', TRAILING BLANKS IN INPUT STRING')  
IF(IM .LT. LA) GOTO 70  
LA = 0  
PAR$ = 0  
IF(IPRT .GE. 20) WRITE(6, 208)  
FORMAT(' FOUND REMAINING STRING IS ALL BLANKS')  
RETURN  
LA = LA - IM  
PAR$ = LA  
IF(IPRT .GE. 20 .AND. LA .EQ. 0) WRITE(6, 209)  
FORMAT(' REMAINING STRING AFTER PARSE FUNCTION IS NULL')  
RETURN  
ERROR HANDLING SECTION FOLLOWS  
WRITE(6, 6001) FUNC$  
FORMAT(' *** STRING LENGTH ERROR *** ', A4)  
PAR$ = -1  
RETURN  
WRITE(6, 6005)  
FORMAT(' *** FATAL ERROR: ATTEMPTED TO PARSE A NULL STRING TWICE')  
STOP  
END
INTEGER FUNCTION POS$(A$, LA$, B$, LB$, LSTART$)

THE STRING A$ IS SEARCHED FOR THE OCCURRENCE OF THE SUBSTRING
B$ AND THE POSITION OF THE 1ST CHARACTER OF A$ WHERE B$ IS
A MATCH IS ASSIGNED TO POS$. THE SEARCH BEGINS AT LSTART$.
IN A$, A ZERO IS RETURNED IF NO MATCH IS FOUND. A NEGATIVE
NUMBER IS RETURNED UPON AN ERROR.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM),B$(IDIM)
INTEGER*4 FUNC$,*POS$ /

IF(IPRT.GE.10) WRITE(6,200)LA$, (A$(I),I=1,LA$)
200 FORMAT(' TRACE A$,I3,,' POS$ =',11A1)
IF(LA$.GT.IDIM.OR.LA$.LT.01 GOTO 6000)
IF(LB$.GT.IDIM.OR.LB$.LT.0) GOTO 6000
C

IF(LB$.LE.(LA$.LSTART+1))GOTO 10
POS$=0
IF(IPRT.GE.20) WRITE(6,201)
201 FORMAT(' FIRST STRING TOO SHORT TO CONTAIN SECOND STRING')
RETURN
C
10
LEFT$=LA$.L8+1
DO 25 1=LEFT$,LSTART$,LEFT$
   DO 20 J=1,LB$
      IF(IPRT.GE.20) WRITE(6,202)IN$,A$(JN$),J,B$(J)
      202 FORMAT(' COMPARE A$(,13,J)=',A$, WITH B$(,13,J)'=',A$)
      IF(A$(JN$).NE.B$(J)) GOTO 25
CONTINUE
POS$=1
IF(IPRT.GE.20) WRITE(6,203) 1
203 FORMAT(' AT POSITION',I3, ' SECOND STRING FOUND IN FIRST')
RETURN
C
C
CONTINUE
ROUTINE WILL FALL THROUGH TO FOLLOWING IF NO MATCH FOUND.
POS$=0
IF(IPRT.GE.20) WRITE(6,204)
204 FORMAT(' FIRST STRING SEARCHED AND SECOND STRING NOT FOUND')
   RETURN
C
C    ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
   POS=-1
   RETURN
   END
INTEGER FUNCTION RCUT$(A$|LA|NUM)

*** STRING A$ HAS NUM CHARACTERS REMOVED FROM THE RIGHT. ***
*** THE VALUE OF THE FUNCTION RCUT$ IS SET TO ***
*** LA IF LA IS GREATER THAN 0 ***
*** 0 IF THE NULL STRING IS LEFT AFTER THE REMOVAL ***
*** -1 IF AN ERROR IS ENCOUNTERED. ***

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 AS(IDIM)
INTEGER*4 FUNC$/'RCUT$/'
INTEGER*4 NUM
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I)|I=1,LA)
200 FFORMAT('TRACE ',I3,'RCUT$ ','I1OAI')
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

LEFT=LA-NUM
IF(LEFT.GT.0) GOTO 20
IF(IPRT.GE.10) WRITE(6,201)LA,(A$(I)|I=1,LA)
201 FFORMAT('STRING NOW ',I4,'','I1OAI')
RCUT$=LA
RETURN

CONTINUE
LA=LEFT
IF(IPRT.GE.20) WRITE(6,201)LA,(A$(I)|I=1,LA)
RCUT$=LA
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$ 6001 FFORMAT('### STRING LENGTH ERROR ### ','A4')
RCUT$=-1
RETURN
END
INTEGER FUNCTION SEG$(A$, LA, B$, LB, LSTAR, LOUT)

** A SUBSTRING IS EXTRACTED FROM A$ AND ASSIGNED TO B$. THE **
** SUBSTRING IN A$ BEGINS AT POSITION LSTAR AND IS LOUT CHAR- **
** ACTERS LONG. **
** IF EITHER A$ IS THE NULL STRING **
** OR A$ DOES NOT HAVE ENOUGH CHARACTERS **
** THEN B$ IS RIGHT PADDED WITH BLANKS SO IT HAS LOUT CHAR **
** IF LOUT=0, THEN B$ BECOMES THE NULL STRING **
** THE ACTUAL NUMBER OF NON-BLANK CHARACTERS OBTAINED FROM A$ **
** IS RETURNED AS THE VALUE OF THE FUNCTION SEG$. **

IMPLICIT INTEGER(A-Z)

COMMON/TEXT/IDIM, IPRT
INTEGER*2 A$(IDIM), B$(IDIM)
INTEGER*2 BLNK/1/
INTEGER*4 FUNC$/'SEG'/

LOGICAL FIRST
IF(IPRT.GE.10) WRITE(6,200)(A$(1), I=1,1,LA)

200 FORMAT('isseur: TRACE ', 'I3, ', SEG$ : 'I1,10A1')
FIRST=.TRUE.
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
IF(LSTART.GT.IDIM.OR.LSTART.LT.0) GOTO 6000
IF(LOUT.GT.IDIM.OR.LOUT.LT.0) GOTO 6000
C
C
IF(LOUT.NE.0) GOTO 5
LB=LOUT
IF(IPRT.GE.20) WRITE(6,202)
202 FORMAT(' NO STRING ASSIGNED BY SEG$')
SEG$=0
RETURN
C
C
5 IF(LSTART.LE.LA) GOTO 15
DO 10 I=1,LOUT
B$(I)=BLNK
10 CONTINUE
LB=LOUT
IF(IPRT.GE.20) WRITE(6,201) LSTART, LOUT
201 FORMAT(' START POINT (I, I3, I) GREATER THAN LENGTH FIRST STRING', SEG$=100
SEG$=SEG$+1
C
C
15 FORMAT(' LSTART GREATER THAN LENGTH OF TEXT ', SEG$=100
SEG$=SEG$+1
C
C
10 FORMAT(' LOUT GREATER THAN LENGTH OF TEXT ', SEG$=100
SEG$=SEG$+1
C
C
50 FORMAT(' TEXT NOT LARGE ENOUGH TO PROVIDE SUBSTRING ', SEG$=100
SEG$=SEG$+1
C
C
100 FORMAT(' FUNCTION SEG$ ', SEG$=100
SEG$=SEG$+1
C
C
1000 FORMAT(' LSTART, LOUT EXCEEDED MAXIMUM ALLOWABLE ', SEG$=100
SEG$=SEG$+1
C
C
10000 FORMAT(' LOUT EXCEEDED MAXIMUM ALLOWABLE ', SEG$=100
SEG$=SEG$+1
C
C
2  /' NEW STRING HAS ',I3,' BLANKS')
   SEG$=0
   RETURN

15  DO 35 I=1,LOUT
    IM=1
    IF(I.GT.(LA-LSTART+1))GOTO 25
    B$(I)=A$(LSTART+I-1)
    GOTO 35
25  B$(I)=BLNK
    IF(.NOT.FIRST)GOTO 35
    FIRST=.FALSE.
    SEG$=IM-1
35  CONTINUE
    IF(FIRST)SEG$=IM
    IF(IPRT.GE.20)WRITE(6,203)SEG$
203  FORMAT(' SEG$ OBTAINED ',I3,' CHARACTERS FROM FIRST STRING')
    LB=LOUT
    RETURN

ERROR HANDLING SECTION FOLLOWS

6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
    SEG$=-1
    RETURN
END
INTEGER FUNCTION TRIM$(A$, LA$)

C*********************************************************************************************
C** THIS FUNCTION STRIPS A STRING OF LEADING AND TRAILING BLANKS.  **C*********************************************************************************************
C** THE RETURN VALUE OF THE FUNCTION IS SET TO THE LENGTH OF THE  **C*********************************************************************************************
C** STRING REMAINING AFTER THE BLANKS ARE REMOVED.  **C*********************************************************************************************
C*********************************************************************************************

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 A$(IDIM), BLNK/
INTEGER*4 FUNCS/"TRIM"/
IF(IPRT.GE.10) WRITE(6,200) LA$(A$(I), I=1, LA$)
200 FORMAT(' TRIM', I3, TRIM$: ', I10A$)
IF(LA$.GT.IDIM . OR. LA$.LT.0) GOTO 6000
C
C
C
CHECK A$ FOR TRAILING BLANKS
C
58 IM=$
   DC 60  I=1, LA$   1
   INDEX=LA$-I+1
   IF(A$(INDEX), NE.BLNK) GOTO 65
   IM=IM+1
   CONTINUE
65 IF(IM.EQ.0) GOTO 70
   IF(IPRT.GE.20) WRITE(6,207) IM
   IF(IM.NE.LA$) GOTO 70
   L$=0
   TRIM$=0
   IF(IPRT.GE.20) WRITE(6,208)
   207 FORMAT(' FOUND', I3, ' TRAILING BLANKS IN STRING')
   208 FORMAT(' FOUND STRING IS ALL BLANKS')
   RETURN
70 IEND=LA$-IM
C
C
CHECK A$ FOR LEADING BLANKS
C
IM=0
DO 10 I=1, IEND
   IF(A$(I), NE.BLNK) GOTO 15
10 CONTINUE
IM = IM + 1
CONTINUE
15 IF (IM.EQ.0) GOTO 25
   IF (IPRT.GE.20) WRITE (6,211) IM
   FORMAT(*) FOUND', I3, ' LEADING BLANKS IN STRING*
211 IM = IM + 1
   IBEG = IBEG + 1

DETERMINE LENGTH OF INPUT STRING
212 LA = IEND - IBEG + 1
IF (LA.LE.IDIM) GOTO 30
   LOSS = LA - IDIM
   IF (IPRT.GE.10) WRITE (6,216) LOSS
   FORMAT(*) STRING TOO LONG FOR MAX STRING LENGTH. LOST',I3
   LA = IDIM
   IEND = IEND - LOSS

TRANSFER THE A$ CHARACTERS TO THE INPUT STRING.
30 INDEX = 1
   DO 85 I = IBEG, IEND
      A$(INDEX) = A$(I)
      IF (IPRT.GE.30) WRITE (6,209) I, A$(I), INDEX, A$(INDEX)
   FORMAT(*) MOVE A$(',I3,')=',A1,' IS NOW A$(',I3,')=',A1
      INDEX = INDEX + 1
85 CONTINUE

CHECK FOR STRING ERROR AND RETURN
   IF (IPRT.GE.10) WRITE (6,222) LA, (A$(I), I = 1, LA)
   FORMAT(*) TRIM$',I3,' AFTER : ',I1) A1
   IF (LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

   TRIM$ = LA
   RETURN

   ERROR HANDLING SECTION FOLLOWS

6000 WRITE (6,6001) FUNC$  
6001 FORMAT(*) *** STRING LENGTH ERROR *** ',A4)
   WRITE (6,6010) LA, LB, IDIM
INTEGER FUNCTION XEQ$(A$, LA, M, M1)

**STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN XEQ INSTRUCTION.**

THE RETURN VALUE OF THE FUNCTION XEQ$ IS SET AS FOLLOWS:

- 0 = CONTINUE TO COMPILE
- -1 = AN ERROR IN COMPILING THE INSTRUCTION.

**IMPLICIT INTEGER(A-Z) COMMON/TEXT/DIM, IPRT COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2 COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2 INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 INTEGER*2 A$(IDIEM)

**INTEGER*2 BLNK /= INTEGER*2 QUOTE/"/ INTEGER*2 IND(3) /'I','A','D'/ INTEGER*2 LABEL(26) /'A','C','H','L','N','S','T'/

**INTEGER*4 FUNC$/'XEQ$/' INTEGER*4 M(11)

IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)

200 FORMAT('TRACE ',13,' XEQ$: ',110A1)

IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

Establish default prefix and instruction length values.

(These are the values for 3 byte local numeric XEQ without IND)

IBYTE=3
PREFIX=224

Strip string of "XEQ" characters.

CALL LCHIT$(A$, LA, 3)

IF(TRIM$(A$, LA)) 6015, 6015, 10

**
CHECK FOR ALPHANUMERIC VERSUS LOCAL LABELS
IF(A$1).EQ.QUOTE) GOTO 80

PROCESS LOCAL LABELS, FIRST CHECK FOR INDIRECT XEQ INSTRUCTION.
P6=POS$(A$,LA,IND,3,1)
IF(P6) 6015,20,16

PROCESS XEQ INDIRECT INSTRUCTION.
P1=P6+3
CALL LCUT$(A$,LA,P1)
IF(IPRT.GE.20)WRITE(6,235)
FORMA'T('DETECTED INDIRECT XEQ INSTRUCTION')
INDIR='TRUE',
I_BYTE=2,
PREFIX=174

CHECK FOR NUMERIC OPERAND
IF(NUMC$(A$,LA,IANSW)) 6015,25,50

OPERAND MUST BE REGISTER X,Y,Z,T OR L OR A LOCAL ALPHA LABEL
DO 30 I=1,26
    INDEX=I
    IF(A$1).EQ.LABEL(I)) GOTO 35
    CONTINUE
30
WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND
GOTO 6015

HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPERAND AND THEN GOTO PROCESS A THREE BYTE INSTRUCTION.

C
35 INDEX=INDEX+101
GOTO 75
C
C
C
C OPERAND MUST BE A NUMERIC LOCAL LABEL
C
50 IF(IVAL$(AS,LA,INDEX))6015,75,75
C
C
C
C PROCESS THE GOTO INSTRUCTION (OPERAND>14)
C
75 M(M)=IBYTE
IF(1PRT.GE.20)WRITE(6,210)M,IBYTE
FORMAT(' XEQ$,15,' LENGTH OF NEXT INSTR IS$,13)
M1=M1+1
C
C
C
C LOAD THE XEQ INSTR PREFIX
C
210 M(M)=PREFIX
IF(1PRT.GE.10)WRITE(6,211)M,M(M)
FORMAT(' XEQ$,15,' XEQ PREFIX INSTR$,T75,I3)
M1=M1+1
C
C
C
C LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTER)
C
211 IF(INDIR)GOTO 95
M(M)=0
IF(1PRT.GE.10)WRITE(6,221)M,M(M)
FORMAT(' XEQ$,15,' NULL FOR XEQ INSTR$,T75,I3)
M1=M1+1
C
C
C
C LOAD THE 2D OPERAND OF THE XEQ INSTR
C
95 IF(INDIR)INDEX=INDEX+128
NOTE THAT FOR XEQ IND THE HIGH ORDER BIT IS SET
M(M)=INDEX
IF(1PRT.GE.10)WRITE(6,212)M,M(M)
C
C
212 FORMAT(' XEQ$",15," XEQ 2D OPERAND ",T75,13)  
M1=M1+1  
XEQ$=0  
RETURN  

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE  
K=0  
P2=POS$(AS,LA,QUOTE,1,2)  
IF(P2) 6015,120,85  

CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS  
LEFT=LA-P2  
IF(LEFT) 6015,100,6015  

DELETE THE ENDING QUOTE BY TRUNCATING THE STRING ONE CHAR  
LA=LA-1  

SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL  
(LINE 120 ACCOUNTS FOR BEGINNING QUOTE STILL ON STRING)  
LENGTH=LA-1  
FOR GTO H=2 FOR LBL H=4 FOR XEQ H=2  
H=2  
I_BYTE=H+LENGTH  
M(M1)=I_BYTE  
IF(IPRT,GE,20)WRITE(6,210)M1,I_BYTE  
M1=M1+1  

HAVE FOUND VALID ALPHANUMERIC LABEL, LOAD "XEQ" INSTRUCTION  
PREFIX=30  
FOR GTO PREFIX=29 FOR LBL PREFIX=192 FOR XEQ PREFIX=30  
XEQ01450  
XEQ01460  
XEQ01470  
XEQ01480  
XEQ01490  
XEQ01500  
XEQ01510  
XEQ01520  
XEQ01530  
XEQ01540  
XEQ01550  
XEQ01560  
XEQ01570  
XEQ01580  
XEQ01590  
XEQ01600  
XEQ01610  
XEQ01620  
XEQ01630  
XEQ01640  
XEQ01650  
XEQ01660  
XEQ01670  
XEQ01680  
XEQ01690  
XEQ01700  
XEQ01710  
XEQ01720  
XEQ01730  
XEQ01740  
XEQ01750  
XEQ01760  
XEQ01770  
XEQ01780  
XEQ01790  
XEQ01800  
XEQ01810  
XEQ01820  
XEQ01830  
XEQ01840  
XEQ01850  
XEQ01860  
XEQ01870  
XEQ01880  
XEQ01890  
XEQ01900  
XEQ01910  
XEQ01920
IF (IPRT, GE, 10) WRITE (6, 214) M1, M(M1)
214 FORMAT (' XEQ$ ', 15, ', ALPHA ', XEQ INSTR$, ', T75, 13)
M1 = M1 + 1
C
C SET INDICATOR FOR NUMBER OF ALPHA CHAR IN LABEL
C U=240
C FOR GTO U=240 FOR LBL U=241 FOR XEQ U=240
M(M1) = U+LENGTH
IF (IPRT, GE, 10) WRITE (6, 216) M1, M(M1)
216 FORMAT (' XEQ$ ', 15, ', LENGTH CODE ALPH ', XEQ $, ', T75, 13)
M1 = M1 + 1
C
C ADD ALPHABETIC CHARACTERS AND RETURN
C LA=LENGTH+1
XEQ$=ALPH$(A$, LA, M, M1)
RETURN
C
C ERROR HANDLING SECTION FOLLOWS
C WRITE (6, 6000) FUNC$
6000 WRITE (6, 6001) FORMAT (' ', *** STRING LENGTH ERROR *** ',', A4)
6001 FORMAT (*** ' ', LA, LB, IDIM)
WRITE (6, 6010) LA, LB, IDIM
6010 FORMAT (LA=' ', I10, ',', LB=' ', I10, ',', IDIM=' ', I10)
XEQ$=-1
RETURN
6015 WRITE (6, 6016) FORMAT (', INVALID SECOND OPERAND IN XEQ INSTR *****)
6016 XEQ$=-1
RETURN
6020 WRITE (6, 6021) FORMAT (', INVALID SECOND OPERAND IN XEQ INSTR *****)
6021 XEQ$=-1
RETURN
END
INTEGER FUNCTION XROS$(AS, LA, M, ML)
C*******************************************************************************
C** STRING AS$ HAS BEEN IDENTIFIED TO CONTAIN AN XROM INSTRUCTION. **XR000020
C** THE XROM INSTRUCTIONS ARE SUBROUTINE CALLS TO HP SUPPLIED ROM ENCODED **XR000030
C** SUBROUTINES. THE SECOND AND THIRD OPERANDS MUST **XR000040
C** BE NUMERIC. **XR000050
C** THE RETURN VALUE OF THE FUNCTION XROS$ IS SET AS FOLLOWS: **XR000060
C** 0 = CONTINUE TO COMPIL **XR000070
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION. **XR000080
C*******************************************************************************
C** IMPLICIT INTEGER(A-Z)
COMMON TEXT/IDIM, [PRT
INTEGER*2 A$(IDIM)
INTEGER*2 COMMA/",/;SS$0(40)
INTEGER*4 FUNC$/"XRO"/
INTEGER*4 M(1)
REAL*4 RFRAC, RDF
IF([PRT*GE.10]) WRITE(6,200)LA; (A$(I),I=1,LA)
200 FORMAT(' TRACES', 'I3', ' XROS$ : '*110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C IF [LCUT$((AS,LA,4)] 6015,6020,7
IF [TRIMS$(AS,LA)] 6015,6020,10
P1=POS$(AS,LA,COMMA,1,1)
P2=P1-1
IF [SEG$((AS,LA,SS$2,1,P2)] 6015,6020,12
IF [LCUT$((AS,LA,P1)] 6015,6020,15
15 IF [IVAL$(SS$, SS$=ROM)] 6030,20,20
IF [IVAL$(AS,LA,PGM)] 6030,25,25
IF [FRAC=ROM/4
IF [FIRST=160+IFRAC
RFRAC=FLOAT(ROM)/.40
RDF=FRAC-FLOT(1IFRAC)
RDF=256.0*RDF
IDF=INT(RDF)
ISECOND=PGM+IDF
C
C SET THE LENGTH OF THE INSTRUCTION
100 IBYTE=2
M(0)=IBYTE
IF(IPRT.GE.20)WRITE(6,215)M(0),IBYTE
215 FORMAT('XRO$,I5,' LENGTH OF THIS INSTR IS$,I3)
M(0)=M(0)+1

C ENCODE THE PREFIX OF THIS INSTRUCTION
M(0)=IFIRST
IF(IPRT.GE.10)WRITE(6,211)M(0),IFIRST,ROM,M(0)
211 FORMAT('XRO$,I5,' PREFIX=',I3,'ROM=',I3,T75,I3)
M(0)=M(0)+1

C ENCODE THE POSTFIX OF THIS INSTRUCTION
M(0)=ISECND
IF(IPRT.GE.10)WRITE(6,221)M(0),ISECND,PGM,M(0)
221 FORMAT('XRO$,I5,' POSTFIX=',I3,'PGM=',I3,T75,I3)
M(0)=M(0)+1

C C

125 XRO$=0
RETURN
C C ERROR HANDLING SECTION FOLLOWS
C 6300 WRITE('XRO$ FUNCTION ',6001) FUNC$
6301 FORMAT('*** STRING LENGTH ERROR *** ',A4)
WRITE(6,6010)LA,IDIM
6010 FORMAT('LA=',I10,'$ IDIM=',I10)
XRO$=-1
RETURN
C 6015 WRITE(6,6016)
6016 FORMAT('**** INVALID ROM NUMBER IN XRO INSTR ****')
XRO$=-1
RETURN
6020 WRITE(6,6021)
6021  FCRMAT(' **** INVALID PROGRAM NUMBER IN XRC INSTR ****')
XR000970
XR000980
XR000990
XR001000
6030  XR001010
XR001020
XR001030
XR001040
6031  FORMAT(' **** NUMERIC CONVERSION ERROR IN XRO INSTR ****')
RETURN
END
This is the program which interfaces the HP41C cross compiler to the Versatec high resolution plotter. If the cross compiler is used on a different plotter, only this routine needs to be changed.

Initialize variables and position pen at origin.

Only two variables (nibs and height) should be changed by the user. Other variables are computed based on these variables.

NIBS is the width of the plotter line (integer 1 to 4). Setting NIBS affects the width of the bar code row. A setting of 4 will give bars of 5 to 6.5 inches in width.

NIBS = 4

HEIGHT is the basic height of the bar code row. Bars may be made in almost any height from 0.2 to 0.5 inches. Recommend a height of 0.40 inches, which will give 12 bars per page.

Hewlett-Packard's bars are .33 inches high.

HEIGHT = 0.40

UNIT is the basic unit width -- it is the width of the space between bars and the width of a zero bar.

UNIT = 0.005 * FLOAT(NIBS)

DOUBLE is twice the width of a unit bar -- -- it is the width of a one bar.

DOUBLE = 0.01 * FLOAT(NIBS)

CSIZE is the height of the bar code row number.

CSIZE = 0.15 * HEIGHT

CSPACE is the height of the space between the bar code row number and the actual bar code row itself.

CSPACE = 0.05 * HEIGHT
CSIZE=0.08*HEIGHT

TSPACE=0.22*HEIGHT

TSPACE IS THE HEIGHT OF THE SPACE BETWEEN THE TITLE AND 
THE FIRST BARCODE ROW NUMBER.

CSIZE+TSPACE+(1.5*CSPACE)

HFACTR IS THE RELATIVE HEIGHT OF THE SUM OF THE BAR HEIGHT

PLUS THE SPACE BETWEEN THE BARS, INCLUDING LABELS

HFACTR=1.7*HEIGHT

PLONG IS THE HEIGHT OF EACH PAGE OF BARCODE

PLONG=1.0

PWIDE IS THE WIDTH OF EACH PAGE OF BARCODE

PWIDE=0.5

TMAR IS THE HEIGHT OF THE TOP MARGIN OF THE PAGE

TMAR=1.0

ESPACE IS THE HEIGHT OF THE MARGINS REPRESENTS HOW 
THE PAPER IS ADVANCED BETWEEN PAGES

ESPACE=TMAR+1.5

SMAR IS THE HEIGHT OF THE LEFT SIDE MARGIN ON THE PAGE

SMAR=1.5

ZERO IS A LEFT MARGIN OFFSET USED TO MOVE THE PHYSICAL 
PLOT AWAY FROM THE LEFT MARGIN OF THE PLOTTER

ZERO=3.0

PERPGE IS THE NUMBER OF BARCODE ROWS THAT WILL BE DRAWN PER PAGE

PERPGE=IFIX((PLONG-ESPACE)/HFACTR)

CALL PLOT(0,0)

CALL PLOT(0.0,0.3)

IROW=0

IPRT=0

X=TMAR

Y=ZERO

READ THE TITLE OF THE BARCODE PROGRAM FROM THE INPUT FILE

READ(5,103)(TITLE(I),I=1,80)

103 FORMAT(80A1)

READ THE ROW OF ZERO'S AND ONE'S FROM THE INPUT FILE

READ(5,101)(END=3000)(IN(I),I=1,132)

101 FORMAT(66A1,/,5X,66A1)

IF(IPRT.GE.20) WRITE(6,201) (IN(I),I=1,132)

201 FORMAT(132A1)
C WRITE THE TITLE ON THE PLOT
C
20 IF(MOD(IROW,PERPGE).NE.0)GOTO 30
X=X+ESPACE
CALL NEWPEN(2)
X1=X+TMAR
X2=X+PLONG-TMAR
Y1=Y-SMAR
Y2=Y+PWIDTH-SMAR
CALL PLOT(X1,Y1,3)
CALL PLOT(X2,Y1,2)
CALL PLOT(X1,Y2,2)
CALL PLOT(X1,Y1,2)
CALL PLOT(X1,Y3)
CALL SYMBOL(X,Y,TSIZE,TITLE,90.0,72)
CALL NEWPEN(NIBS)
X=X+TSPACE
Y=ZERO
CALL PLOT(X,Y,3)

C
223

C LABEL THE BAR CODE ROW
C
30 ROWNUM=ROWNUM+1.0
IROW=IROW+1
CALL NEWPEN(2)
CALL NUMBER(X,Y,CSIZE,ROWNUM,90.0,-1)
CALL NEWPEN(NIBS)
X=X+CSPACE
CALL PLOT(X,Y,3)

C

C CONVERT THE ZERO'S AND ONE'S INTO BARS OF CORRECT WIDTH
C
DO 1000 I=1,132
C
C
C
50 CHECK FOR ZERO OR ONE BIT
IF(INI.EQ.ZERO)GOTO 133
IF(INI.EQ.ONE)GOTO 230
IF(INI.EQ.BLNK)GOTO 2000
C
C     DRAW A ZERO BAR OF UNIT WIDTH
  100   X=X+HEIGHT
       CALL PLOT(X,Y,2)
       Y=Y+DOUBLE
       X=X-HEIGHT
       CALL PLOT(X,Y,3)
       GOTO 1000

C     DRAW A ONE BAR OF DOUBLE UNIT WIDTH
 200   X=X+HEIGHT
       CALL PLOT(X,Y,2)
       Y=Y+UNIT
       X=X-HEIGHT
       CALL PLOT(X,Y,3)
       X=X+HEIGHT
       CALL PLOT(X,Y,2)
       Y=Y+DOUBLE
       X=X-HEIGHT
       CALL PLOT(X,Y,3)
       GOTO 1000

1000  CONTINUE

       GOTO NEXT ROW
 2000   X=X+FACTR
       Y=ZERO
       CALL PLOT(X,Y,3)
       GOTO 10

C     FINISH UP THE PLOT: PROGRAM IS COMPLETE
 3000   CALL PLOT(0.,0.,+999)
       STOP
       END
THE FOLLOWING IS THE DATA SET READ BY THE CROSS COMPILED.

<table>
<thead>
<tr>
<th>HP41C SOURCE CCDE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>- 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>/ 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>* 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>6 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>5 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>4 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>3 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>2 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>1 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>0 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>HP41C SOURCE CCDE:</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>+ 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>- 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>/ 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>* 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>6 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>5 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>4 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>3 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>2 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>1 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>0 2 2 2 2 2 2 2</td>
</tr>
</tbody>
</table>

235
LIST OF REFERENCES

1. Wickes, W. C., Synthetic Programming on the HP41C, Larkin Publications, P.O. Box 987, College Park, Maryland 20740, 1980.


<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>2</td>
<td>Defense Technical Information Center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cameron Station</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alexandria, Virginia 22314</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>Library, Code 0142</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>3.</td>
<td>1</td>
<td>Department Chairman, Code 55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Department of Operations Research</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>4.</td>
<td>5</td>
<td>Associate Professor S. H. Parry, Code 55Py</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Department of Operations Research</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>5.</td>
<td>5</td>
<td>Associate Professor R. H. Shudde, Code 55Su</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Department of Operations Research</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>6.</td>
<td>1</td>
<td>Office of the Commanding General</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U. S. Readiness Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: General Donn A. Starry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MacDill AFB, Florida 33621</td>
</tr>
<tr>
<td>7.</td>
<td>1</td>
<td>Office of the Commanding General</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U. S. Army Training and Doctrine Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: General Glenn Otis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fort Monroe, Virginia 23651</td>
</tr>
<tr>
<td>8.</td>
<td>1</td>
<td>Chief</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRADOC Research Element Monterey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>9.</td>
<td>1</td>
<td>Headquarters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U. S. Army Training and Doctrine Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: Director, Studies and Analysis Directorate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mr. S. Goldberg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fort Monroe, Virginia 23651</td>
</tr>
<tr>
<td>10.</td>
<td>1</td>
<td>Headquarters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U. S. Army Training and Doctrine Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: ATCG-T (BG Morelli)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fort Monroe, Virginia 23651</td>
</tr>
</tbody>
</table>
11. Headquarters
U. S. Army Training and Doctrine Command
ATTN: Director, Maneuver Directorate Combat Developments (LTC John VanZant)
Fort Monroe, Virginia 23651

12. Headquarters
U. S. Army Training and Doctrine Command
ATTN: ATGN-AE (LTC Smith)
Fort Monroe, Virginia 23651

13. Mr. Walter Hollis
Deputy Under Secretary of the Army
(Operations Research)
Department of the Army
Washington, D.C. 20310

14. LTG Howard Stone
Commanding General
U.S. Army Combined Arms Center
Fort Leavenworth, Kansas 66027

15. Director
Combined Arms Combat Development Activity
ATTN: ATZL-CAC-A (Mr. Lee Pleger)
Fort Leavenworth, Kansas 66027

16. Director, BSSD
Combined Arms Combat Development Activity
ATTN: ATZLCA-DS
Fort Leavenworth, Kansas 66027

17. Director
Combat Analysis Office
ATTN: Mr. Kent Pickett
Fort Leavenworth, Kansas 66027

18. Commandant
U.S. Army Command and General Staff College
ATTN: Education Advisor
Room 123, Bell Hall
Fort Leavenworth, Kansas 66027

18. Commandant
U.S. Army Command and General Staff College
ATTN: ATZL-SWB (MAJ Witskonke)
Fort Leavenworth, Kansas 66027

19. Dr. Wilbur Payne, Director
U.S. Army TRADOC Systems Analysis Activity
White Sands Missile Range, New Mexico 88002

20. Headquarters, Department of the Army
Office of the Deputy Chief of Staff
for Operations and Plans
ATTN: DAMO-2D
Washington, D.C. 20310
21. Commander
U. S. Army Concepts Analysis Agency
ATTN: MOCA-WG (LTC Earl Jarden)
8120 Woodmont Avenue
Bethesda, Maryland 20014

22. Director
U. S. Army Material Systems Analysis Activity
ATTN: DRSY-CM (Mr. Bill Niemeyer)
Aberdeen Proving Grounds, Maryland 21005

23. Director
U. S. Army Night Vision and Electro-Optical Lab
ATTN: DEL-NV-VI (Mr. Frank Shields)
Fort Belvoir, Virginia 22060

24. Director
U. S. Army TRADOC Systems Analysis Activity
ATTN: Mr Ray Heath
White Sands Missile Range, New Mexico 88002

25. Director
Combat Developments Studies Division
ATTN: MAJ W. Scott Wallace
U. S. Army Armor Agency
Fort Knox, Kentucky 40121

26. Commandant
U. S. Army Field Artillery School
ATTN: ATSF-MBT (CPT Steve Starner)
Fort Sill, Oklahoma 73503

27. Director
Combat Developments
U. S. Army Aviation Agency
Fort Rucker, Alabama 36362

28. Director
Combat Developments
U. S. Army Infantry School
Fort Benning, Georgia 31905

29. Director
Armored Combat Vehicle Technology Program
ATTN: COL Fleming
U. S. Army Armor Center
Fort Knox, Kentucky 40121

30. Director
Combat Developments
ATTN: MAJ William D. Meiers
U. S. Army Air Defense Agency
Fort Bliss, Texas 79905

31. Commander
U. S. Army Logistics Center
ATTN: ATCL-OS (Mr. Cameron/CPT Schuessler)
Fort Lee, Virginia 23801
32. Director
U. S. Army Material Systems Analysis Agency
ATTN: DRISY-AA (Mr. Tom Coyle)
Aberdeen Proving Grounds, Maryland 21005

33. Deputy Chief of Staff for Combat Developments
U. S. Army Training and Doctrine Command
ATTN: (MG Dick Boyle)
Fort Monroe, Virginia 23651

34. Deputy Commanding General
Combined Arms Combat Development Activity
ATTN: ATZL-CA-DC
Fort Leavenworth, Kansas 66027

35. Commandant
U. S. Army Signal School
Fort Gordon, Georgia 30905

36. Associate Professor Arthur L. Schoenstadt, Code 53Zh
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940

37. Associate Professor James K. Hartman, Code 55Hh
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

38. Professor James G. Taylor, Code 55Tw
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

39. Professor Gary K. Poock, Code 55Pk
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

40. Professor Donald R. Barr, Code 55Bn
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

41. Mr. Henry Horn, Editor
Hewlett-Packard Keynotes
Hewlett-Packard Corporation
1000 W. E. Circle Blvd.
Corvallis, Oregon 97330

42. Mr. Richard Nelson, Editor
PCC Journal
2541 W. Camden Place
Santa Ana, California 92704
43. Headquarters, Department of the Army
Office of the Deputy Chief of Staff for Personnel
ATTN: DAPE-MPE-SS (Captain James Richmann)
Washington, D. C. 20310