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

SEP 81 J N RICHMANN
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**

James Norman Richmann

Naval Postgraduate School
Monterey, California 93940

Monterey, California 93940

Unclassified

Approved for public release; distribution unlimited.

**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
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.
# 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
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Programming Environment Command Menu</td>
<td>12</td>
</tr>
<tr>
<td>2.</td>
<td>List of Commands</td>
<td>13</td>
</tr>
<tr>
<td>3.</td>
<td>On-Line Introductory Material</td>
<td>15</td>
</tr>
<tr>
<td>4.</td>
<td>Components of Program Documentation</td>
<td>21</td>
</tr>
<tr>
<td>5.</td>
<td>Example Program to Add n Numbers</td>
<td>30</td>
</tr>
<tr>
<td>6.</td>
<td>Program to Recall an Element of a Matrix</td>
<td>34</td>
</tr>
</tbody>
</table>
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 documentation on how to use the system and what commands and options are available. Figure 1 shows the command menu displayed on the terminal screen by this interactive program:
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 HP411 EMULATOR</td>
</tr>
<tr>
<td>PP23</td>
<td>COMP</td>
<td>C</td>
<td>COMPILE 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></td>
<td>CMS</td>
<td></td>
<td>ALLOWS EXECUTION OF ANY VALID CMS COMMAND</td>
</tr>
<tr>
<td></td>
<td>CP</td>
<td></td>
<td>ALLOWS EXECUTION OF ANY VALID CP COMMAND</td>
</tr>
</tbody>
</table>

INPUT COMMAND:

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 TOTALLY 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 MVS 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 INITIATE NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH PILETYPE &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>PP19 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>PP20 OCOMP</td>
<td>C O</td>
<td>THIS COMMAND IS USED TO PRODUCE AN &quot;OFFLINE&quot; COMPILATE. THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPILATE WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.</td>
</tr>
<tr>
<td>PP21 PRINT</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>PP22 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>PP23 COMP</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 COMPILATE THE USER IS AUTOMATICALLY PLACED IN THE CMS BROWSE MODE FOR THE OUTPUT &quot;LISTING&quot; FILE THAT RESULTED FROM THE COMPILATE.</td>
</tr>
<tr>
<td>PP24 XEDIT</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
   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 zero's and
   one's 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 per-
   form this step until 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 algorithm 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 clear 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</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Establishes a loop counter.</td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Clears ( x ) and pushes loop counter into ( y ).</td>
</tr>
<tr>
<td>LBL 00</td>
<td></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 insure 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 indirect 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 differenti-
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 cannot 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 * (ADDRESS IS NOW IN X REG
9 + (ADDRESS IS NOW IN X REG
10 RCL INV 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 programming. It would be able to increase the modularity of programs, 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 subroutines include a complete set of string functions for manipulating 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 constructed 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 components of the proposed calculator programming support system would work together efficiently.
APPENDIX A

SINGLE VARIABLE STATISTICS EXAMPLE

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 "S1" 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:

\[
\begin{array}{ccccccc}
2.1 & 2.4 & 2.2 & 2.7 & 2.5 \\
2.4 & 2.6 & 2.6 & 2.3 & 2.9 \\
\end{array}
\]

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:

   \[ \text{XEQ ALPH\text{A SIZE ALPH\text{A 26}} \]

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

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

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:

\[ \text{XBAR} = 2.470 \]

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:

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

Note that if flag 21 is set on (press \( \text{SF 21} \)), 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 About the Mean</td>
</tr>
<tr>
<td>SX=0.241</td>
<td>Sample Standard Deviation</td>
</tr>
<tr>
<td>SKEW=0.170</td>
<td>Skewness</td>
</tr>
<tr>
<td>KURTO=2.302</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.
<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 nnn</td>
<td>UP TO nnn = 235</td>
<td></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 R/S</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ENTER THE DATA X1?,X2? ETC.</td>
<td>input R/S</td>
<td></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>SSQD=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.</td>
<td>PRGM</td>
<td>STANDBY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AND THEN DISPLAY:</td>
<td>MED=XX</td>
<td>R/S</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></td>
<td></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>
<tr>
<td>STEP</td>
<td>EXPLANATION</td>
<td>SEE</td>
<td>PRESS</td>
<td>RESULT</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-----</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>7</td>
<td>USER OPTION TO ENTER NUMBER OF HISTOGRAM CELLS. NO INPUT IS REQUIRED.</td>
<td>CELL?</td>
<td>R/S &quot;-OF-&quot; INPUT N</td>
<td>R/S</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 &quot;-OR-&quot; INPUT</td>
<td>R/S</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 FREQ COUNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XX=xx</td>
<td>R/S</td>
<td>UPPER X-VALUE LIMIT</td>
</tr>
<tr>
<td>10</td>
<td>ACCEPT NEXT COMMAND</td>
<td>CMD</td>
<td>ENTER NEXT CMD</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>RECALCULATE HISTOGRAM</td>
<td></td>
<td>&quot;XEQ &quot;AGAIN</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>EDIT AN INPUT VALUE AT ANY TIME PRIOR TO DATA SORT.</td>
<td></td>
<td>&quot;XEQ &quot;SC</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></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>13</td>
<td>REVIEW DATA POINTS (OR REVIEW ORDER STATS AFTER SORT.)</td>
<td></td>
<td>XEQ</td>
<td>SR</td>
</tr>
</tbody>
</table>
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
(-------------------------------------------)
<table>
<thead>
<tr>
<th>STAT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 LBL &quot;STAT1&quot; RECOMMENDED ENTRY POINT</td>
</tr>
<tr>
<td>2 CF 05 SET VERIFY MODE OFF</td>
</tr>
<tr>
<td>3 SP 26 ENABLE AUDIO</td>
</tr>
<tr>
<td>4 GTO &quot;SS&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 LBL &quot;S1&quot; ENTRY POINT FOR CLASSROOM USE</td>
</tr>
<tr>
<td>6 SP 05 SET VERIFY MODE ON</td>
</tr>
<tr>
<td>7 CF 26 DISABLE AUDIO TONES</td>
</tr>
<tr>
<td>8 SP 21 SET TO STOP DURING VERIFICATION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&quot;SS&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 LBL &quot;SS&quot; ENTRY POINT FOR USER SET OPTIONS</td>
</tr>
<tr>
<td>10 CF 29 NO DIGIT GROUPING</td>
</tr>
<tr>
<td>11 REG 10 ESTABLISH STATISTICAL REGISTERS</td>
</tr>
<tr>
<td>12 CF 06 USED BY DATA REVIEW FUNCTION</td>
</tr>
<tr>
<td>13 CF 08 USED BY DATA EDITING FUNCTION</td>
</tr>
</tbody>
</table>

| 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 PROMPT CLEAR MEANS DATA ENTRY, NOT REVIEW |
| 19 ? (CLEAR MEANS DATA ENTRY, NOT REVIEW     |
| 20 FC06 PROMPT 06 ? (CLEAR MEANS DATA ENTRY, NOT REVIEW |
| 21 PROMPT 06                   |
| 22 / (CLEAR MEANS DATA ENTRY, NOT REVIEW     |
| 23 1 (CLEAR MEANS DATA ENTRY, NOT REVIEW     |
| 24 + (CLEAR MEANS DATA ENTRY, NOT REVIEW     |
| 25 STO 01 (SET UP LOOP COUNTER FOR DATA POINTS |

58
```
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

DATA ENTRY LOOP

01

28 LBL 01
29 ISG 00
30 LBL 02
31 RCL IND 00
32 FIX 0
33 ARCL 01
34 FIX 3
35 LSTO 03
36 FS? 06
37 GTO 03
38 R+=?
39 TONE 9
40 PROMPT
41 STO IND 00
42 FC? 05
43 GTO 04
44 STO 03
45 CLA
46 LBL 03
47 ARCL 03
48 ARCL IND 00
49 CP 22
50 VIEW
51 FC? 22
52 GTO 04
53 STO IND 00
54 GTO 03
55 LBL 04
56 ST+ 10
57 SA 11
58 LSTX
59 ST+ 12
60 LASTX
61 ST+ 13
62 LASTX
63 ST+ 13
64 LASTX
65 FS? 08
66 GTO 01
67 ISG 01
68 RCL 01
69 GTO 04
70 RTN
71 ISG 01
72 GTO 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 \{STOR ES THE NUMBER OF DATA POINTS\}
75 MEAN
76 "XBAR
77 XEQ 97 \{CALL AN OUTPUT LABELING ROUTINE\}
78 STO 03 \{TEMP STORE FOR XBAR\}
79 RCL 11 \{RECALL SIGMA X-SQUARED\}
80 RCL 03 \{RECALL XBAR\}
81 X \^ 2
82 RCL 15 \{RECALL NUMBER POINTS\}
83 RCL 09 \{TEMP STORE FOR SUM OF SQUARED\}
84 \^ \{DEVATIONS ABOUT THE MEAN\}
85 STO 09 \{NUMBER POINTS\}
86 "SSQD
87 XEQ 97 \{CAN NOT USE SDEV FUNCTION BECAUSE OF\}
88 RCL 15 \{NON-STANDARD USE OF REGISTERS 12-14\}
89 \^ \{STANDARD DEVIATION\}
90 1
91 / \{SUM OF SQ DEVIATION ABOUT MEAN\}
92 \{NUMBER POINTS\}
93 XEQ 97 \{SECOND MOMENT\}
94 RCL 09 \{SIGMA X-CUBED\}
95 RCL 15 \{SIGMA X-SQUARED\}
96 RCL 12 \{XBAR\}
97 RCL 11 \{NUMBER POINTS\}
98 RCL 03 \{XBAR\}
99 X \^ 3
100 - \{THIRD MOMENT\}
101 RCL 05 \{SECOND MOMENT\}
102 \{SIGMA X-FOURTH-POWER\}
103 X \^ 3
104 + \{XBAR\}
105 RCL 15 \{NUMBER POINTS\}
106 RCL 03 \{XBAR\}
107 RCL 03 \{XBAR\}
108 \{SIGMA X-FOURTH-POWER\}
109 3
110 \Y \^ 3
111 + \{XBAR\}
112 \Y \^ 3
113 2
114 + \{THIRD MOMENT\}
115 STO 06 \{SECOND MOMENT\}
116 RCL 05 \{SIGMA X-POURTH-POWER\}
117 RCL 13 \{SIGMA X-CUBED\}
118 RCL 03 \{XBAR\}
119 "SKEW
120 XEQ 97 \{OUTPUT THE SKEWNESS OF THE DATA\}
121 RCL 13 \{SIGMA X-POURTH-POWER\}
122 RCL 12 \{SIGMA X-CUBED\}
123 RCL 03 \{XBAR\}

60
HP41C SOURCE CODE:  SINGLE VARIABLE STATISTICS

125  \*  
126  4  \*  
128  \* \ RCL 03  
130  \* \ RCL 11  
132  \* \ 6  \*  
134  \* \ RCL 15  
136  \* \ RCL 03  
138  \* \ RCL \( \bar{x} \)  
140  \* \ RCL \( \Sigma x^2 \)  
142  \* \ RCL \( \Sigma x \)  
144  \* \ RCL \( \Sigma x \)  
146  \* \ RCL \( \Sigma x \)  
148  \* \ RCL \( \Sigma x \)  
150  \* \ RCL \( \Sigma x \)  
152  \* \ RCL \( \Sigma x \)  
154  \* \ RCL \( \Sigma x \)  
156  \* \ RCL \( \Sigma x \)  
158  \* \ RCL \( \Sigma x \)  
160  \* \ RCL \( \Sigma x \)  
162  \* \ RCL \( \Sigma x \)  
164  \* \ RCL \( \Sigma x \)  
166  \* \ RCL \( \Sigma x \)  
168  \* \ RCL \( \Sigma x \)  
170  \* \ RCL \( \Sigma x \)  
172  \* \ RCL \( \Sigma x \)  
174  \* \ RCL \( \Sigma x \)  

(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  

61
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

{ "AGAIN" DISPLAY HISTOGRAM }

175 LBL "AGAIN" (ADDRESS BASE REGISTER
176 RCL 04 (RECALL THE FIRST ORDER STAT
177 1 (CALL AN OUTPUT LABELING ROUTINE
178 + (HOLDS STARTING (LEFTMOST) X BOUNDARY
179 RCL IND X (ADDRESS BASE REGISTER
180 MIN (NUMBER OF DATA POINTS
181 XEQ 97 (RECALL THE N-TH ORDER STATISTIC
182 STO 09 (DISPLAY THE MAX VALUE OBSERVED
183 RCL 04 (ADDRESS BASE REGISTER
184 + (MIN
185 RCL IND X (ADDRESS BASE REGISTER
186 MAX (TEMP STORE FOR THE RANGE
187 XEQ 97 (DISPLAY THE RANGE
188 RCL 04 (INITIALIZE TEMP FLAG TO MARK LAST BAR
189 RCL 15 (ADDRESS BASE REGISTER
190 + (NUMBER DATA POINTS
191 RCL 15 (COMPUTING INDEX LOOP COUNTER
192 RCL 04 (ADDRESS BASE REGISTER
193 + (ADDRESS AND LOOP THRU DATA
194 1 (NUMBER POINTS
195 LN (DEFAULT NUMBER OF BARS IS 2*LN(N)
196 RCL 04 (VALUE IS ROUNDED NOT TRUNCATED
197 IE3 (USER HAS OPTION TO CHANGE NUMBER CELLS
198 RCL 04 (RANGE
199 IE3 (USER HAS OPTION TO CHANGE CELL WIDTH
200 RCL 04 (NOW HOLDS CELL WIDTH NOT RANGE
201 + (CELL WIDTH
202 RCL 04 (UPPER LIMIT OF CURRENT CELL COUNTED
203 + (INITIALIZE CELL COUNTER
204 RCL 01 'NEXT DATA POINT
205 RCL 09 'ICELL (CELL UPPER LIMIT
206 PIXEL 3 (NEXT DATA POINT
207 X<>Y (USER HAS OPTION TO CHANGE CELL WIDTH
208 RND (NOW HOLDS CELL WIDTH NOT RANGE
209 RND (CELL WIDTH
210 RND (UPPER LIMIT OF CURRENT CELL COUNTED
211 RND (INITIALIZE CELL COUNTER
212 RND (NEXT DATA POINT
213 RND (CELL UPPER LIMIT

62
HP41C SOURCE CODE:

SINGLE VARIABLE STATISTICS

<table>
<thead>
<tr>
<th>Line No.</th>
<th>HP41C Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>229</td>
<td>X&lt;Y?</td>
<td>(Data point less than upper limit)</td>
</tr>
<tr>
<td>230</td>
<td>GTO 08</td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>232</td>
<td>ST+ 02</td>
<td>(Increment the cell counter)</td>
</tr>
<tr>
<td>233</td>
<td>ISG 01</td>
<td>(Prepare to look at next data point)</td>
</tr>
<tr>
<td>234</td>
<td>GTO 07</td>
<td>(Set flag for output of last bar)</td>
</tr>
<tr>
<td>235</td>
<td>SF 00</td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>LBL 08</td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>&quot;CNT&quot;</td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>RCL 02</td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>FIX 0</td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>XEQ 97</td>
<td>(Output the cell frequency count)</td>
</tr>
<tr>
<td>241</td>
<td>FIX 3</td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>&quot;XX&quot;</td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>RCL 09</td>
<td>(Output cell boundary—lower limit)</td>
</tr>
<tr>
<td>244</td>
<td>XEQ 97</td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>FC?C 00</td>
<td>(Is this the last bar?)</td>
</tr>
<tr>
<td>246</td>
<td>GTO 06</td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>&quot;CHD&quot;</td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>AVIEW</td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>RTN</td>
<td></td>
</tr>
</tbody>
</table>
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

Review the data

```
250 LBL "SR
251 SF 06
252 GTO 00

(set mode for review not query)
```

Edit the data

```
253 LBL "SC
254 SF 08
255 CP 06
256 RCL 00
257 STO 05
258 RCL 01
259 STO 06

(set to edit mode)
(set to query for correct pointer)
(current input address pointer)
(save to enable return to data entry)
(current input index loop counter)
(save to enable return to data entry)

"point?
261 PROMPT
262 STO 01
263 RCL 04

(establish pseudo-index counter)
(address base register)
(computer address of data to be edited)
(recall the old value)
(correct sigma x)
(correct sigma x-squared)
(correct sigma x-cubed)
(correct sigma x-fourth-power)
(call data entry as a subroutine)
(following restores data entry)
(input address register value)
(input loop counter)

264 * STO 00

(test to see if all data already input)

265 RCL IND 00
266 ST- 10
267 X^2
268 ST- 11

270 LASTX
271 *
272 ST- 12
273 LASTX
274 * ST- 13
275 XEQ 02
276 CP 02
277 RCL 05
278 STO 00
279 RCL 06
280 STO 01

282 1
283 -
284 ISG X
285 GTO 02
286 GTO "5M
```

64
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

OUTPUT LABELING ROUTINE

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

98 SHELL SORT

292 LBL 98
293 RCL 15
294 STO 01
295 2
296 ÷
297 INT
298 ÷
299 STO 01
300 RTN
301 X=0?
302 RTN
303 1
304 STO 02
305 LBL 10
306 STO 03
307 RCL 01
308 ÷
309 RCL 04
310 +
311 ÷
312 RCL IND X
313 +
314 RCL 04
315 ÷
316 X<>Y
317 LBL 12
318 X<>Y
319 X<>Y
320 X<>Y
321 X<>Y
322 LBL 13
323 RCL 01
324 ÷
325 STO 03
326 X=0?
327 STO 10
328 LBL 11
329 RCL 01
330 ÷
331 LBL 01
332 ÷
333 RCL 02
334 ÷
335 ÷
336 STO 02
337 X<>Y
338 STO 09
THIS PROGRAM USES THE FOLLOWING REGISTERS:

R00 -- INPUT DATA ADDRESS POINTER
R01 -- LOOP INDEX COUNTER
R02 -- TEMP REGISTER
R03 -- TEMP REGISTER
R04 -- INDIRECT ADDRESS BASE
R05 -- SECOND MOMENT (POPULATION VARIANCE)
R06 -- THIRD MOMENT
R07 -- FOURTH MOMENT
R08 -- HISTOGRAM CELL WIDTH
R09 -- TEMP REGISTER
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
SINGLE VARIABLE STATISTICS
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 "REDA" 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 "REDA." 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:

\[
A + B \leq 30
\]
\[
A + B + C + D \leq 50
\]
\[
B + C \geq 30
\]
\[
D \geq 5
\]

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>3</td>
<td>C</td>
<td>D</td>
<td>H1</td>
<td>H2</td>
<td>S1</td>
<td>S2</td>
<td>A1</td>
<td>A2</td>
<td>RHS</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-4</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</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:

```
XEQ ALPHA SIZE ALPHA 175
```

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

```
XEQ ALPHA LP 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 \text{ 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 \text{ 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:

\begin{center}
\begin{tabular}{|l|c|}
\hline
\text{See} & \text{Respond With} \\
\hline
Basic 2 ? & 8 \text{ R/S} \\
Basic 3 ? & 9 \text{ CHS R/S} \\
Basic 4 ? & 10 \text{ CHS R/S} \\
\hline
\end{tabular}
\end{center}

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 SF 04 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:

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:

`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>
</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>
</tr>
<tr>
<td>B</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>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td></td>
<td></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 ((NNN=21M+(N+1)(M+2))) where (M=NUM) (RW)s and (N=NUM) (C)ols</td>
<td>XEQ &quot;SIZE</td>
<td>NNN</td>
<td>Up to (N) (NNN) = (177)</td>
</tr>
<tr>
<td>2</td>
<td>Call the program</td>
<td>XEQ &quot;LP&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Enter the number of constraints</td>
<td>NUM_RGW</td>
<td>INPUT M</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Enter the number of variables (Include slacks, surplus &amp; artif.)</td>
<td>NUM_COL</td>
<td>INPUT N</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Enter current basic variable numbers by row</td>
<td>BASIC 1 ?</td>
<td>INPUT VAR #</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Enter tableau values. For mistakes or to review the data see last step below</td>
<td>T1,1? ETC.</td>
<td>INPUT R/S</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>To force the calculator to stop after each pivot</td>
<td>R/S</td>
<td>SF 04</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Simplex completed: Optimal solution found</td>
<td>VALUE= XX.XXX</td>
<td>Final Obj. Function Value</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Simplex completed: Problem is infeasible</td>
<td>INFEAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>EXPLANATION</td>
<td>SEE</td>
<td>PRESS</td>
<td>RESULT</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-----</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>10</td>
<td>SIMPLEX COMPLETED: PROBLEM IS UNBOUNDED.</td>
<td>UNBOUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TO REVIEW VALUES IN TABLEAU AT ANY TIME INCLUDING FINAL TABLEAU.</td>
<td></td>
<td>XEq &quot;ALP&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AS PROMPTS ApPEAR, DATA CAN BE CHANGED BY ENTERING NEW VALUE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHAT IS CURRENTLY BASIC?</td>
<td>BASIC</td>
<td>CLX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHAT ARE VALUES IN TABLEAU?</td>
<td>CLX</td>
<td>PROMPT WILL VANISH LEAVING DATA</td>
<td></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>

85
HP41C SOURCE CODE:  LINEAR PROGRAMMING

```
{ "LP
 { "LP
 { "LP

1 LBL "L1P
2 XEQ "READMN
3 XEQ "READJB
4 XEQ "READA
5 XEQ "CHECK
6 LBL 15
7 XEQ "PINDQ
8 RCL 05
9 XEQ "READN
10 FS? 11
11 GTO 35
12 XEQ "REALA
13 "CHECK
14 LB
15 XEQ "PIDQ
16 RCL 05
17 RCL 10
18 RCL 12
19 X* +
20 RCL 07
21 RCL IND Y
22 ABS
23 X<Y?
24 GTO 20
25 "IMPEAS
26 "AVIZEW
27 STOP
28 LBL 20
29 SF 11
30 RCL 04
31 RCL 08
32 RCL 12
33 X* +
34 STO 15
35 GTO 15
36 LBL 25
37 RCL 04
38 RCL 09
39 X* +
40 RCL IND X
41 STO 00
42 "VALUE=
43 ARCL 00
44 AVIEW
45 STOP
46 LBL 35
47 XEQ "PINDP
48 XEQ "EBINDP
49 XEQ "EBINDP
50 GTO 40
51 "UNBOUND
```
HP41C SOURCE CODE: LINEAR PROGRAMMING

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

FINDQ

60 LBL "FINDQ
61 SF 01
62 STO 03
63 STO 05
64 RCL 11
65 XEQ 'SETL
66 LBL 31
67 RCL 31
68 RCL 01
69 +
70 STO 00
71 RCL IND X
72 RCL 03
73 -
74 RCL 07
75 ABS
76 X<;I?
77 GTO 38
78 FC? 11
79 GTO 37
80 RCL 01
81 RCL IND X
82 ABS
83 RCL 07
84 X<;I?
85 GTO 38
86 LBL 37
87 RCL IND 00
88 STO 03
89 RCL 01
90 RST
91 STO 05
92 LBL 38
93 ISG 01
94 GTO 31
95 CF 01
96 RIN

88
HP41C SOURCE CODE: LINEAR PROGRAMMING

```
{ 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 + STO 00
116 RCL 05
117 + RCL IND X
118 + RCL 00
119 RCL 12
120 + RCL IND X
121 STO 02
122 RCL 07
123 GTO 48
124 RCL 00
125 RCL 12
126 + RCL IND X
127 RCL 02
128 STO 03
129 RCL 03
130 - RCL 07
131 RCL CHS
132 XEQ?
133 GTO 48
134 LBL 47
135 STO 06
136 RCL 01
137 INTP
138 GTO 01
139 RCL 48
140 GTO 00
141 RCL 03
142 RCL IND X
143 STO 06
144 LBL 48
145 STG 01
146 GTO 41
147 CF 02
148 RTN
```
HP41C SOURCE CODE: LINEAR PROGRAMMING

PIVOT

150 LBL "PIVOT"
151 SF 03
152 RCL 06
153 1
154 RCL 12
155 +
156 RCL 04
157 +
158 STO 03
159 RCL 05
160 +
161 RCL IND X
162 1/X
163 STO 00
164 RCL 12
165 \(X^2\)
166 XEQ "SETL"
167 LBL 51
168 RCL 03
169 RCL 01
170 +
171 RCL 00
172 STO IND Y
173 ISG 01
174 RCL 51
175 1
176 RCL 09
177 FS? 11
178 PC? 11
179 1
180 +
181 XEQ "SETL"
182 LBL 52
183 RCL 06
184 RCL 01
185 1
186 RCL IND X
187 1
188 "SETL"
189 STO 59
190 1
191 RCL 12
192 +
193 RCL 04
194 +
195 1
196 STO 16
197 RCL 05
198 +
199 RCL IND X
200 STO 00
201 RCL 12
202 XEQ "SETL"
HP41C SOURCE CODE: LINEAR PROGRAMMING

204  LBL 53
205  CF 07
206  RCL 05
207  RCL 02
208  INT
209  X=Y?
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
225  LBL 54
226  TSG 02
227  GTO 53
228  LBL 59
229  TSG 01
230  GTO 52
231  RCL 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  "PRCHPT"
243  "PRCHPT"
244  "PRCHPT"
245  "PRCHPT"
HP41C SOURCE CODE: LINEAR PROGRAMMING

{ READMN }

LBL "READMN"
246 70 STO 00
247 16  "NUM ROWS
248 13  "NXT
249 18  "NXT
250 17  "NXT
251 7 "NUM COLS
252 "NXT
253 10 RCL
254 16 RCL 04
255 17 +
256 01 STO + 00
257 00 STO IND 00
258 09 RCL
259 04 +
260 16 "SIZE?
261 25 PC'C
262 25 PROMPT
263 25 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

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

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

LBL "SETL"
284 LBL "SETL"
285 1E03
286 /
287 1
288 STO IND Y
289 RTN
```

93
HP 41C SOURCE CODE: LINEAR PROGRAMMING

READJB

291 LBL "READJB"
292 1 RCL 08
293 XEQ "SETL"
294 RCL 13
295 STO 00
297 FIX 0
298 LBL 01
299 "BASIC"
300 ARCL 01
301 "="
302 ARCL "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 2
314 RCL 12
315 XEQ "SETL
316 LBL 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 *+ RCL 02
328 + RCL 04
329 + 1
330 - STO 00
331 LBL 14
332 PC? 10
333 GTO 16
334 XEQ "IN
335 GTO 17
336 LBL 16
337 =
338 ARCL IND 00
339 AVIEW
340 LBL 17
341 ISG 02
342 GTO 12
343 ISG 01
344 GTO 11
345 C? 10
346 RTN
A CROSS COMPILER AND PROGRAMMING SUPPORT SYSTEM FOR THE HP41CV — ETC(U)
HP41C SOURCE CODE:  LINEAR PROGRAMMING

{(-----------------------------)
{| CHECK |
|-------------------------------|
{(-----------------------------)

LBL "CHECK
SF 11
RCL 12
XEQ "SETL
RCL 09
RCL 12
RCL 01
STO 14
STO 15
LBL 91
RCL 14
CL 01
STO IND Y
ISG 01
GTO 91
1
RCL 08
XEQ "SETL
LBL 92
RCL 07
RCL 13
RCL 01
RCL IND X
RCL 00
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X=Y?
GTO "ERR1
RCL 09
XEQ "SETL
LBL 93
RCL 08
RCL 01
INT
RCL 02
2XM?
HP41C SOURCE CODE: LINEAR PROGRAMMING

428 RCL 04
429 RCL 04
430 +
431 FC? 08
432 0
433 FS? 08
434 1
435 STO IND Y
436 ISG 02
437 GTO 93
438 FC? 07
439 GTO 98
440 CF 11
441 BCL 14
442 RCL 00
443 +
444 STO IND Y
445 STO IND Y
446 RCL 12
447 XEQ \#SETL
448 LBL \#SETL
449 RCL 01
450 INT
451 1
452 -
453 RCL 12
454 RCL 02
455 X<>Y
456 RCL IND X
457 +
458 RCL IND Y
459 +
460 STO IND Y
461 ISG 02
462 GTO 93
463 LBL 96
464 ISG 02
465 XEQ \#SETL
466 LBL \#SETL
467 0
468 -
469 RCL 04
470 RCL 04
471 RCL 04
472 +
473 RCL 12
474 RCL 02
475 X<>Y
476 RCL IND Y
477 +
478 STO IND Y
479 ISG 02
480 GTO 93
481 LBL 98
482 ISG 01
483 GTO 92
484 FC? 11
485 RTN
486 RCL 04
487 RCL 04
488 *+
489 RCL 12
490 RCL 12
491 PC? 11
492 RTN
493 *+
494 RCL 14
495 RCL 14
496 RTN
497 *+
498 RCL 14
499 RTN

98
THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG ASSIGNMENTS MADE BY THIS PROGRAM:

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

**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 preceeded 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 LBL "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 algorithm
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

P41C SOURCE CODE: COMMUN SUBROUTINES

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" {INSURE NO QUERY
5 CF 10 {ASSUMES NO POINTS TO STORAGE REG.
6 LBL "IX" {DIRECT MODE--ASSUMES X REG HOLDS DATA
7 RCL IND 00 {ASSUMES LABEL SET UP BY CALLING PRG
8 ASTC 05 {QUERY OR DISPLAY VALUE?
9 LBL 01 {QUERY OR DISPLAY VALUE?
10 LBL 02 {QUERY OR DISPLAY VALUE?
11 LBL 03 {QUERY OR DISPLAY VALUE?
12 FS? 10 {QUERY OR DISPLAY VALUE?
13 FS? 10 {QUERY OR DISPLAY VALUE?
14 FS? 10 {QUERY OR DISPLAY VALUE?
15 ARCL IND 00 {QUERY OR DISPLAY VALUE?
16 CF 22 {QUERY OR DISPLAY VALUE?
17 PRCPRT {QUERY OR DISPLAY VALUE?
18 PRCPRT {QUERY OR DISPLAY VALUE?
19 CLA {QUERY OR DISPLAY VALUE?
20 FC? 05 {QUERY OR DISPLAY VALUE?
21 GTI 03 {QUERY OR DISPLAY VALUE?
22 LBL "L" {INPUT MODE--DISPLAY LABEL AND ?
23 FS? 22 {QUERY OR DISPLAY VALUE?
24 GTI 10 {QUERY OR DISPLAY VALUE?
25 GTI 04 {QUERY OR DISPLAY VALUE?
26 LBL "L" {QUERY OR DISPLAY VALUE?
27 CF 10 {QUERY OR DISPLAY VALUE?
28 ARCL 05 {QUERY OR DISPLAY VALUE?
29 GTO 02 {QUERY OR DISPLAY VALUE?
30 LBL 00 {QUERY OR DISPLAY VALUE?
31 FS? 10 {QUERY OR DISPLAY VALUE?
32 GTO 04 {QUERY OR DISPLAY VALUE?
33 FS? 22 {QUERY OR DISPLAY VALUE?
34 GTI 02 {QUERY OR DISPLAY VALUE?
35 LBL 04 {QUERY OR DISPLAY VALUE?
36 ISG 00 {QUERY OR DISPLAY VALUE?
37 RTN {QUERY OR DISPLAY VALUE?
38 RTN {QUERY OR DISPLAY VALUE?

116
SOURCE CODE:  COMMON SUBROUTINES

D2

36 LdL "02  (SETUP LOOP USING REG 2 AS INDEX COUNTER
40 2
41 GTC 07  (SETUP LOOP USING REG 1 AS INDEX COUNTER
42 L3L "01
43 1
44 L3L 07  (NUMBER LOOP ITERATIONS MUST BE IN X
45 X>Y  PRIOR TO CALLING THIS SUBROUTINE.
46 1E3
47 / 1
48 +
50 STC IND Y
51 RTN
FP41C SOURCE CODE: COMMON SUBROUTINES

```assembly
VR

52 LSL "VR
53 SF C5
54 LBL "WR
55 CF Z6
56 LBL "MS
57 CLRG
58 &REG 10
59 CF Z9
60 IE-C4
61 STC 03
62 JS 13
63 STD 04
64 ?I
65 STO 00
   (PRT: ADV
   (PRT: SF L2
   (PRT: FC? 55
   (PRT: PRA
   (PRT: CF 12
   (PRT: 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 STORE).
(DOUBLE WIDE PRINTING
(PRINTER ATTACHED?)
(SET BACK TO SINGLE WIDE PRINT
(HAS INITIALIZATION ROUTINE BEEN ASKED TO
VERIFY SIZE?)
```
HP41C SOURCE CODE: COMMON SUBROUTINES

```
70 LBL "SZ
71 "SET SZ=
72 ARCL X
73 1
74 SF 25 (PREPARE TO IGNORE ERROR
75 SF 25 (TEST FLAG 25 TO SEE IF SUFFICIENT
76 RCL IND X 25 TO SEE IF SUFFICIENT
77 FC?C 25
78 PRCTPT
79 RTN (SIZE EXISTS.

80 LBL "ER
81 0 (DISPLAY "DATA ERROR" PROMPT & SOUND TCNE.
82 LN (BEST WAY TO DETERMINE WHERE ERROR
83 TONE 2 (OCCURED IS TO HIT THE SST KEY ONCE,
84 RTN (THEN GO INTO PRGM MODE.
```
HP41C SOURCE CODE: COMMON SUBROUTINES

LBL "SCRT"

STC 01

RCL 15 (RECALL NUMBER OF DATA POINTS)

RCL C1 (RECALL MIDPOINT)

STC 01

LBL C2 (TEST TO SEE IF LIST SORTED)

RCL 01

STC 01

LBL 00

STC 01

LBL 02

RTN

STC 02

LBL 01

STACK TABLE FOLLOWS:

C = B

RCL 01

RCL 02

RCL 04

RCL 03

RCL 05

RCL 04

RCL 05

RCL 06

RCL 07

RCL 08

RCL 09

RCL 10

RCL 11

RCL 12

RCL 13

RCL 14

RCL 15

RCL 16

RCL 17

RCL 18

RCL 19

RCL 20

RCL 21

RCL 22

RCL 23

RCL 24

RCL 25

RCL 26

RCL 27

RCL 28

RCL 29

RCL 30

RCL 31

SORT USES THE FOLLOWING REGISTERS:

R01 — A

R02 — B

R03 — C

R04 — INDIRECT ADDRESS BASE

R15 — NUMBER DATA POINTS (SET BY &+1)
I-P41C SOURCE CODE: COMMON SUBROUTINES

```
{(-----
  {   SC
   }
  ----})

133 LBL "SC
134 RCL 00
135 1
136 -
137 STC 05
138 -IEC3
139 STC 10
140 -1
141 3
143 STC 09
```
FF4LC SOURCE CODE: COMMON SUBROUTINES

PLT

LBL "PUT"
MVALLE
STC 00
XEQ "MIA"
SF C2

GET

LBL "GET"
MBIT REG
PRMPRT
CCCMON SUBROUTINES

{ ---------- }  
|           |  
|           |  
|           |  
|           |  
| SA        |  
|           |  
{ ---------- }

153       LBL "SA  
154       3  
155       /  
156       INT  
157       STC 07  
158       LASTX  
159       RCL 09  
160       /  
161       12  
162       +  
163       STO 03  
164       RCL 03  
165       ST+ 07  
166       RCL IND 07  
167       INT  
168       STC 1+  
169       LASTX  
170       -  
171       RCL 10  
172       *  
173       INT  
174       STO 13  
175       LASTX  
176       -  
177       RCL 10  
178       *  
179       INT  
180       STC 12  
181       RCL 11  
182       RCL IND 08  
183       STO 11  
184       FC TC 02  
185       RTN  
186       X<Y  
187       STC IND 08  
188       RCL 12  
189       RCL 10  
190       CHS  
191       /  
192       RCL 13  
193       +  
194       RCL 10  
195       CHS  
196       /  
197       RCL 14  
198       +  
199       STC IND 07  
200       RTN  

123
THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG ASSIGNMENTS MADE BY THIS PROGRAM:

- **R00** = INDIRECT ADDRESS FOR STORAGE OF INPUT DATA
- **R01** = LOOP INDEX COUNTER
- **R02** = LOOP INDEX COUNTER
- **R03** = EFFECTIVE ZERO LEVEL -- USE AS TEMP IF NA
- **R04** = BASE REGISTER FOR INDIRECT ADDRESSING (15)
- **R05** = TEMP REGISTER FOR ALPHA PROMPT
- **R06** = **R09** = APPLICATION PROGRAM TEMP REGISTERS
- **R10** = **R15** = STATISTICAL REGISTERS -- USE AS TEMP IF NA
- **R16** = STORAGE OF DATA VIA INDIRECT ADDRESSING

**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 CALC. 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 JIDE PRINTING -- LOCAL WITHIN "IC"
- **F13** = WHEN SET MEANS MAIN ROUTINE ASKING "VR" FOR AUTOMATIC SIZE CHECK AFTER INITIALIZATION.
- **F14** = **F20** = AVAILABLE FOR APPLICATION PROGRAM USE.
COMMON SUBROUTINES
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 subrou-
tines. 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
 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.
# 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 Execs is that for ASCII terminals the printing of the command menu is suppressed 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
```
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 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
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 ZERO'S AND
ONE'S 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 PER-
FORM THIS STEP UNTIL YOUR PROGRAM HAS SUCCESSFULLY
COMPILe WITHOUT ERRORS. THIS STEP IS DONE BY THE
BATCH PROCESSOR AND IT 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 "II" 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" FACILITY FROM "FLIST" SIMPLY TYPE "PF19" IN THE COMMAND AREA.

NOW TO BEGIN:

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

- NEW &BEGTYPE -ENDQQ ENTER CMS FILENAME OF YOUR PROGRAM..... (PF13 OR "STOP" RETURN TO CMS)
- ENDQQ
- SWITC#1 = ON &READ ARGS &GOTO -CHECK &PROGRAM
&PROCNAME = &1
&PRTYPE = HP41
&PROGMODE = A1
&SWITCH1 = OFF
STATE &PROCNAME &PRTYPE &PROGMODE
&IF &RETCODE = 0 &GOTO -DISPLAY
CLRSCRN
&GRTYPE = -ENDINTRO2

ENTER THE LABEL YOU WISH TO HAVE PRINTED AT THE TOP OF THE BARCODE.
-ENDINTRO2
&READ ARGs
&STACK I &1 &2 &3 &4 &5 &6 &7 &8 &9 &10
&STACK LBL &PROCNAME
&STACK SET TABS 1 20 25 20 35 45 55 55
&STACK SET TRUNC 57
&STACK I
XEDIT &PROCNAME &PRTYPE &PROGMODE

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

-DISPLAY
&IF /&SWITCH1 = /ON &GOTO -NEW
&IF /ASCII = /YES &GOTO -ENDDISP
CLRSCRN
&TYPE HP41C CROSS COMPILER .......... &PROCNAME .......... EDITION=17 SEP 81
&GRTYPE = -ENDDISP

SELECT DESIRED COMMAND FROM THE FOLLOWING:

PF-KEY  COMMAND  CODE  ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM
PF13    STOP     S      GETS YOU OUT OF THE HP41C 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 HP41C PROGRAMS ON DISK
PF20    OLOMP    C,O    OFFLINE COMPIL AND AUTO GENERATION OF BAR CODE
PF21    PRINT    P      PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM
PF22    *        *      RESERVED FOR FUTURE USE BY HP41 EMULATOR
PF23    COMP     C      COMPILE A SOURCE LISTING ON CMS DISK
PF24    XEDIT    X      EDIT THE PROGRAM USING THE CMS FULL-SCREEN EDITOR
ERASE  ERASE THE SOURCE FILE, LISTING FILE AND TEXT FILE
CMS    ALLOWS EXECUTION OF ANY VALID CMS COMMAND
CP

ALLOWS EXECUTION OF ANY VALID CP COMMAND

-ENDDISP

*TYPE INPUT COMMAND:

*READ ARGS

CLRSCRN

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

***

** 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 -LISTFIL

*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 -LISTFIL

*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 /D = /61 &GOTO -DISPLAY

*IF /P = /61 &GOTO -TYPE

*IF /C = /61 &GOTO -INNER

*IF /Q = /61 &GOTO -OCOMP

*IF /C = /61 &GOTO -COMPILE

*IF /X = /61 &GOTO -XEDIT

*IF /G1 = /FILEDEF &GOTO -INNER
GIF /G1 = /DATA &GOTO -INNER
GIF /ESWITCH = /ON &GOTO -PROGNAME
CTYPE 27 &L &E 43 64 65 UNRECOGNIZED
-ENDCHECK &GOTO -ENDDISP

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

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

CMSCMD &A = &RANGE OF & 2 &N
ETRACE CN
ETRACE COMMAND &A
ETRACE OFF
&GOTO -ENDDISP
-CPCMD &A = &RANGE OF & 2 &N
ETRACE CN
CP &A
ETRACE OFF
&GOTO -ENDDISP
-XEDIT
ESTACK SET TABS 1 20 25 35 45 55 65
ESTACK SET TRUNC 57
-XEDIT &PROGNAME &PROGTYPE &PROGMEMODE
&GOTO -DISPLAY

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

** LIST HP41C PROGRAM FILES ON CMS DISK **

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

-LISTFILE
FLIST * HP41 A
&GOTO -DISPLAY

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

** COMPIL **

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

-COMPILE
FILEDEFF 05 DISK &PROGNAME &PROGTYPE
FILEDEFF 06 DISK &PROGNAME LISTING
FILEDEFF 04 DISK &PROGNAME DATA
FILEDEFF 02 DISK INSTR CODES &USERMODE
CTYPE CROSS-COMPILE BEGINS.
-HPCCROSS
-BROWSE &PROGNAME LISTING
&GOTO -DISPLAY

******************************************************************************
** OFFLINE COMPIL**

- **OCMP**
- ERASE PROGRAM DATA
- FILEDEF 05 DISK &PROGNAME &PROTYPE
- FILEDEF 06 DISK &PROGNAME LISTING
- FILEDEF 04 DISK &PROGNAME DATA
- FILEDEF 02 DISK INSTR CODES &USERMODE

** TYPE CROSS-COMPIL BEGIN**

** HPCROSS **

PRINT PROGRAM LISTING (UP)

** STATE PROGRAM DATA **

** IF @RETCODE = 0 &GOTO -SUBMIT **

** TYPE COMPIL OF PROGRAM WAS NOT SUCCESSFUL. **

** GOTO -ENDDISP **

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

** **

** -ENTER **

** &BEGIN -ENDCAUTION **

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 ENTIRE PROGRAM. FOR THESE REASONS YOU MAY WISH
TO EDIT A SOURCE CODE FILE FIRST, AND THEN SUBMIT THIS FILE
FOR CROSS-COMPIILATION WITH THE "COMP" COMMAND.

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

** INPUT RESPONSE: **

- ** -ENDCAUTION **
- ** GREAD ARGS **
- ** IF /A = /Y &GOTO -DISPLAY **
- ** CLEAR **
- ** 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 &PROGNAME DATA **
- ** FILEDEF 02 DISK INSTR CODES &USERMODE **
- ** HPCROSS **
- ** GOTO -ENDDISP **
*** DISPLAY A MESSAGE THAT FUNCTION IS NOT AVAILABLE. ***

**NOTYET
CLRSCRN
&EGTYPE -ENDYET

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

THANK YOU.
-ENDDISP

** TYPE LISTING FILE

**

**

** TYPE
PRINT &PROGNAME LISTING IUP
&GOTO -DISPLAY

** ERASE SOURCE, LISTING AND TEXT FILES

**

** ERASE
ERASE &PROGNAME LISTING
ERASE &PROGNAME DATA
ERASE LCAO MAP

&TYPE WARNING: DO YOU WISH TO ERASE THE SOURCE CODE? [YES/NO]?

&READ VARS &ANSW

&IF &ANSW = /YES &GOTO -DISPLAY

ERASE &PROGNAME &PROCTYPE &PROGMODE
&GOTO -DISPLAY

**

** SUBMIT TO MVS FOR BATCH PROCESSING

**

** SUBMIT
-TRYSUBMIT
&PERM = SUBMIT
&EGTYPE -ENDSUBM

ENTER JOB NAME AND USERID:
-ENDDSUBM
READ VARS &JN &USERID
&JN = &PIECE OF &JN 1 8
&USERID = &PIECE OF &USERID 1 4
&STACK I //&JN JOB &USERID 'I0111' 'HP41CV BAR CODE',CLASS=A
&STACK C //I0111',I0111/'
&STACK I // EXEC FRXCLGF
&STACK I // FORT,SYSIN DD *
&STACK GET VERA FORTRAN &USERMODE
&STACK I //GO, PLOTPARM DD *
&STACK GET PLOTPARM JCL &USERMODE
&STACK I //GO,SYSIN DD *
&STACK GET &PROGNAME DATA
&STACK I /*
&STACK FILE
XEDIT &PROGNAME JCL
EXEC SUBMIT &PROGNAME JCL
ERASE &PROGNAME JCL
&GOTO -DISPLAY

**********************************************************************************
** PRINT INSTRUCTIONS
**
***
***
***
-HELP
&BECTYPE -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
HP41C PROGRAMS AND RETURN TO CMS. IF YOU ARE EXECUTING A FUNCTION THAT WAS INVOKED FROM THE COMMAND MENU, IN MOST CASES PF13 WILL RETURN YOU TO THE MENU, AND BY PRESSING PF13 AGAIN YOU WILL RETURN TO CMS.

PF14 HELP H 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.

PF15 ENTER E 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 TOTALLY RESPONSIBLE FOR UPPER AND LOWER CASE BEING ENTERED PROPERLY.

PF16 BAR B THIS COMMAND IS USED ONCE THE HP41C PROGRAM IS WRITTEN AND COMPILED WITHOUT ERRORS. IT SUBMITS A JOB TO MVS BATCH FOR THE PHYSICAL PRODUCTION OF THE BAR CODE.

PF17 NEW N THIS COMMAND IS USED TO DIRECT THE ATTENTION OF THE COMMAND PROCESSOR TO A NEW HP41 PROGRAM SOURCE FILE. WHEN USED TO INITIATE NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH FILETYPE "HP41" AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.

PF18 DIREC D 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 "INPUT COMMAND".

PF19 LIST L THIS COMMAND DISPLAYS "FLIST" 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.

PF20 CCOMP C O THIS COMMAND IS USED TO PRODUCE AN "OFFLINE" COMPIL. THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPIL WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.
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-COMPILE\'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 EMULATION 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 COMPILE THE USER IS AUTOMATICALLY PLACED IN THE CMS BROWSE MODE FOR THE OUTPUT "LISTING" FILE THAT RESULTED FROM THE COMPIL.

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 PROponent FOR THE HP41C SYSTEM.

-ENUHELP CGO TC -ENDDISP

** ****************************************** **
** EXIT COMMAND PROCESSOR **
**
** ****************************************** **
-EXIT CP SPOOL PRINTER CLOSE NOCONT
CLRSCHRN &EXIT
DONE=.FALSE.
READ(2,101) (PRT,PAGE
READ(2,9101) (TITLE$(JJ),JJ=1,25)
FORMAT(25AI)
DO 5 I=1,NIINST
READ(2,1021)(INSTR$,(J,I),J=1,6),CCOE(I),LINST(I)
CONTINUE
READ(5,103) (TITLE$(I),I=26,IDIM)
FORMAT(75AI)
LENGTH=O
DO 16 I=1,MAX
   (I=THE INSTR NUMBER; J=CHARACTER IN INSTR)
ATTEMPT TO READ A TEXT STRING.
   IF(IPRT,GE,20)WRITE(6,292) UNDER
   FORMAT('T','NEXT INSTRUCTION:',15A4,/
   IF(INST$(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. ***
HPC00490
HPC00500
HPC00510
HPC00520
HPC00530
HPC00540
HPC00550
HPC00560
HPC00570
HPC00580
HPC00590
HPC00600
HPC00610
HPC00620
HPC00630
HPC00640
HPC00650
HPC00660
HPC00670
HPC00680
HPC00690
HPC00700
HPC00710
HPC00720
HPC00730
HPC00740
HPC00750
HPC00760
HPC00770
HPC00780
HPC00790
HPC00800
HPC00810
HPC00820
HPC00830
HPC00840
HPC00850
HPC00860
HPC00870
HPC00880
HPC00890
HPC00900
HPC00910
HPC00920
HPC00930
HPC00940
HPC00950
HPC00960
C 12 IF(T$(1).NE.COMENT)GOTO 13
   IF(MOD(LINCNT,PAGE).EQ.0.OR.(T$(2).EQ.COMENT.AND.LT.GE.2)
      CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE$)
      LINCNT=LINCNT+1
      WRITE(6,268) (T$(J),J=1,LT)
      FORMAT(110A1)
      IF(IPRT.GE.10)WRITE(6,263)
      FORMAT(4 FOUND COMMENT CARD. NOTHING MORE DUNE."
      GOTO 16
C C C IF NOT A COMMENT, INCREMENT THE INSTRUCTION COUNTER AND PRINT
 THE INSTRUCTION.
C C LENGTH=LENGTH+1
   IF(MOD(LINCNT,PAGE).EQ.0)
      CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE$)
      LINCNT=LINCNT+1
      WRITE(6,269)LENGTH,(T$(J),J=1,LT)
      FORMAT(110A1)
C C TRIM OFF TRAILING COMMENTS
C C IF(FIND$(COMENT,1,T$,LT,LOC)) 6000,9915,9914
C C 9914 LT=LOC-1
C C 9916 IF(TRAN$(T$,LT)) 6000,9916,9915
C C CONTINUE
C C 9915 CONTINUE
C C COMPILE THE TEXT INSTRUCTION.
C C IF(COMP$(T$,LT,M,M1)) 15,16,20
C C 15 ERROR=.TRUE.
C C 16 CONTINUE
C C GOTO THE FOLLOWING INSTRUCTIONS IF END OF FILE ENCOUNTERED
C C 14 WRITE(6,259)
C C 259 FORMAT(1 END OF FILE**********HPC00970
C C HPC00990
C C HPC10090
C C HPC10100
C C HPC00120
C C HPC00130
C C HPC00140
C C HPC00150
C C HPC00160
C C HPC00170
C C HPC00180
C C HPC00190
C C HPC00200
C C HPC00210
C C HPC00220
C C HPC00230
C C HPC00240
C C HPC00250
C C HPC00260
C C HPC00270
C C HPC00280
C C HPC00290
C C HPC00300
C C HPC00310
C C HPC00320
C C HPC00330
C C HPC00340
C C HPC00350
C C HPC00360
C C HPC00370
C C HPC00380
C C HPC00390
C C HPC00400
C C HPC00410
C C HPC00420
C C HPC00430
C C HPC00440
CALL THE BAR CODE GENERATOR.

\begin{verbatim}
30     MSAVE=M1
       B1=1
       IBAR=8STR$(M,M1,ITOT,TITLE$)
       WRITE(6,301) ITOT
301    FORMAT(1 END HP41C CROSS COMPIL'15,' BYTES IN TOTAL PROGRAM'
     HPC01590
     HPC01600
     HPC01610
     HPC01620
     HPC01630
     HPC01640
     HPC01650
     HPC01660
     HPC01670
     HPC01680
     HPC01690
     HPC01700
     HPC01710
     HPC01720
     HPC01730
     HPC01740
     HPC01750
     HPC01760
     HPC01770
     HPC01780
\end{verbatim}
INTEGER FUNCTION AIN$(INOPER,B)
210 D(2) = 1
OPERND = D2

CONVERT THE THIRD BINARY DIGIT

230 D(3) = OPERND - 32
IF(D(3) 300, 310, 310
300 B(3) = 0
GOTO 340
310 B(3) = 1
OPERND = D3

CONVERT THE FOURTH BINARY DIGIT

340 D(4) = OPERND - 16
IF(D(4) 400, 410, 410
400 B(4) = 0
GOTO 450
410 B(4) = 1
OPERND = D4

CONVERT THE FIFTH BINARY DIGIT

450 D(5) = OPERND - 8
IF(D(5) 500, 510, 510
500 B(5) = 0
GOTO 560
510 B(5) = 1
OPERND = D5

CONVERT THE SIXTH BINARY DIGIT

560 D(6) = OPERND - 4
IF(D(6) 600, 610, 610
600 B(6) = 0
GOTO 670
610 B(6) = 1
OPERND = D6

CONVERT THE SEVENTH BINARY DIGIT

670 D(7) = OPERND - 2
IF(D(7) 700, 710, 710

700  B(7)=0
    GOTO 780
710  B(7)=1
    OPERNO=D7
C
C   CONVERT THE EIGHTH  BINARY DIGIT
C
780  D8=OPERNO-1
800  B(I8)=D8
     IF(B(I8)) 800,810,6000
810  B(I8)=0
     GOTO 1000
C
C   WRITE OUT CONVERSION IF NECESSARY AND RETURN
C
1000 IF(IPRT.GE.20) WRITE(6,66) INOPER,B(1:8)
     FORMAT('TRACEx AINS OPERAND='B(1:15),BINARY='B(8)' )
     AINS=0
     RETURN
C
C   ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001  FORMAT('***** CONVERSION ERROR ***** A4')
     WRITE(6,6002) INOPER
     FORMAT('ERROR IN AINS OPERAND='.15)
     AINS=-1
     RETURN
   END
I3=1
M(1)+C2(I3)
IF(IPRT,GE,10)WRITE(6,208)M1,C$(I3),C2(I3),M(1)

2 FORMAT(' ALPHS',I5,' ALPHA CHARACTER ',75,I3)

I3=I3+1
CONTINUE

C C C
40 ALPHS=0
RETURN
C C
ERROR HANDLING SECTION FOLLOWS
C C
6000 WRITE(6,6001) FUNCS
6001 FORMAT(' *** STRING LENGTH ERROR *** ','A4')
WRITE(6,6010) LA,LB,IDIM
6010 FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
ALPHS=-1
RETURN
END
INTEGER FUNCTION ALPI$(A$, LA, M, M1) ALP00010
C***************************************************************ALP00020
C** STRING A$ BEGINS WITH A QUOTE AND HENCE IS AN ALPHA ENTRY ALP00030
C** INSTRUCTION. THIS ROUTINE COMPiles SUCH INSTRUCTIONS. ALP00040
C** THE RETURN VALUE OF THE FUNCTION ALPI$ IS SET AS FOLLOWS: ALP00050
C** 0 = CONTINUE TO COMPile ALP00060
C** -1 = AN ERROR IN COMPiling THE INSTRUCTION. ALP00070
C*****************************************************************ALP00080
C
IMPLICIT INTEGER(A-Z) ALP00100
COMMON/TEXT/IDIM,IPRT ALP00110
COMMON/FLAGS/DONE,POIGIT,ALPHA,INDIR,FLAG1,FLAG2 ALP00120
COMMON/CNTR/PI,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 ALP00130
LOGICAL DONE,POIGIT,ALPHA,INDIR,FLAG1,FLAG2 ALP00140
INTEGER*4 PI,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2 ALP00150
INTEGER*2 A$(IDIM) ALP00160
INTEGER*2 BLNK/* */ ALP00170
INTEGER*2 QUOTE/* */.APPEND/* - */ ALP00180
INTEGER*4 FUNC/* ALPI$/ ALP00190
INTEGER*4 M(I) ALP00200
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA) ALP00210
200 FORMAT(TRACE *,I3,' ALPI$',I*,I0A1) ALP00220
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000 ALP00230
C
C STRIP OFF THE LEADING QUOTE ALP00240
C IF(LENGTH$(A$,LA,1)) 6015,6015,10 ALP00250
C C
C STRIP OFF THE TRAILING QUOTE, IF ANY ALP00260
C IF(A$(LA).EQ.QUOTE|LA=LA-1 ALP00270
C C
C SET THE LENGTH OF THE INSTRUCTION ALP00280
C IF(LA.LT.15) GOTO 15 ALP00290
C WRITE(6,204) ALP00300
204 FORMAT('**** ALPHA STRING TOO LONG *****') ALP00310
C LA=15 ALP00320
15  IBYTE=LA+1
   M(MI)=IBYTE
   IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215  FORMAT(' ALP1$',15,' LENGTH OF THIS INSTR IS',I3)
   M1=M1+1
   ENCODE THE TEXT LENGTH INSTRUCTION
C
   M(MI)=240+LA
   IF(IPRT.GE.10)WRITE(6,211)M1,M(MI)
211  FORMAT(' ALP1$',15,' TEXT LENGTH INSTR',T75,13)
   M1=M1+1
   CHECK FOR ALPHA APPEND INSTRUCTION
C
   IF(A$(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(' ALP1$',15,' ALPHA APPEND CHAR',T75,13)
   M1=M1+1
   IF(LCUT$(A$,LA,1)) 6015,6015,50
C
   ENCODE TEXTUAL STRING
C
   ALP1$=ALPH$(A$,LA,P,M1)
   RETURN
   ERROR HANDLING SECTION FOLLOWS
C
   6000  WRITE(6,6001) FNC$
   6001  FORMAT(' *** STRING LENGTH ERROR *** ','A4)
      WRITE(6,6010) LA,LB,IDIM
   6010  FORMAT(' LA=','110,' LB=','110,' IDIM=','110)
      ALP1$=-1
      RETURN
   6015  WRITE(6,6016)
   6016  FORMAT(' **** INVALID OPERAND IN ALPHA ENTRY INSTR **** ')
      ALP1$=-1
      RETURN
   END
INTEGER FUNCTION ASGN$(A$,L$A$,B$,LB$)

C** THIS FUNCTION IS A STRING ASSIGNMENT OPERATOR. THE STRING
C** IN A$ IS COPIED INTO B$, THE NULL STRING LA=0 IS A VALID
C** STRING AND WILL BE COPIED CORRECTLY.
C**
C** COMMON/TEXT/IDIM, IPRT
C** INTEGER*2 A$(IDIM), B$(IDIM)
C** INTEGER*4 FUNC$, 'ASGN$'
C** IF(IPRT.GE.10) WRITE(6,200)LA$(I),A$(I),I=1,LA$)
C** 200 FORMAT(' 'TRACE ',I3,' ASGN$ ; ',110A$')
C** IF(LA$.GT.IDIM.OR.LA$.LT.0) GOTO 6000
C C
C IF(LA$.EQ.0) GOTO 20
C 10 DC 15 I=1,LA$
C B$(I)=A$(I)
C 15 CONTINUE
C 20 LB=LA$
C ASGN$=1
C RETURN
C C
C ERROR HANDLING SECTION FOLLOWS
C
C 6000 WRITE(6,6001) FUNC$
C 6001 FORMAT(*** STRING LENGTH ERROR *** ,A4$
C WRITE(6,610) LA$,LB,LDIM
C 610 FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
C ASGN$=0
C RETURN
C END

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

C******************************************************************************
C**
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 COMPILC
C** -1 = AN ERROR IN COMPIIING THE INSTRUCTION.
C**
C******************************************************************************

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 W(133), TITLE$(133)
INTEGER*2 ALPHA([33], BLNK=' ', ZERO='0', ONE='1'
INTEGER*4 Funcs, BSTR$
IM(1), M1, Wl
IF (IPRT .GE. 10) WRITE(6,200)
   200 FCRMAT('TRACE ',13,' BSTR$ : ','

C**
C** INITIALIZE COUNTERS

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

C**
C** WRITE THE TITLE TO THE BINARY CODE ARRAY

C WRITE(4,776)(TITLE$(JJ), JJ=26, IDIM)
C 776 FORMAT(80A1)

C**
C** CHECK FOR END OF PROGRAM

C 320 IF (M(P) .LE. -99) GOTO 530

C******************************************************************************
C  EXTRACT NUMBER OF BYTES IN INSTRUCTION
  IBYTE=M(P)
  NBYTE=IBYTE
  P=P+1
C
C  EXTRACT NEXT OPERAND OF THE INSTRUCTION
  OPNED=M(P)
  P=P+1
C
C  CONVERT OPERAND TO BINARY AND LOAD INTO ARRAY W
  CHECK=CHECK+OPNED
  IF(CHECK.GT.255) CHECK=CHECK-255
  IF(IPRT.GE.10) WRITE(6,555)ROW,OPNED,CHECK
  555 FORMAT(5, SEND TO AINS ROW=*,13,* OPERAND=*,16,* CHECKSUM=*,15)
    IF(AINS(OPERND,W(WI))) 6000,420,420
C
C  IF SUCCESSFUL CONVERSION, DECREASE BYTES REMAINING
C  INCREMENT THE ROW COUNT, AND CHECK TO SEE IF END OF BARCODE ROW
  IBYTE=IBYTE-1
  WI=WI+8
  ROW=ROW+1
  IF(ROW.EQ.13) GOTO 530
C
C
C  CHECK TO SEE IF INSTRUCTION HAS BEEN COMPLETELY ENCODED
  IF(IBYTE.EQ.0) GOTO 520
  GOTO 390
C
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
C
PROGRAM, THEN CHECK FOR CONTINUATION OF INSTRUCTION THAT MUST CROSS BARCODE BOUNDARIES.

530     WP=W1
     IF(1BYTE.LE.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
SINCE THE HP=41C INSTRUCTIONS ARE OF VARYING LENGTH, THEY WILL
MOST COMMONLY STRADDLE THE BORDER BETWEEN TWO ROWS OF BAR CODE.
THE THIRD BYTE OF A BAR-CODE ROW CONTAINS, IN THE 4 HIGH ORDER
BITS, THE NUMBER OF LEADING BYTES AND IN THE 4 LOW ORDER BITS
THE NUMBER OF TRAILING BYTES.

580     THIRD=(16*LEAD)+TRAIL
     W1=19
     CHECK=CHECK+THIRD
     IF(PIR,T.GT.255) CHECK=CHECK-255
     IF(PIR,T.GE.10) WRITE(6,551)ROW,THIRD,CHECK
     IF(INSTHIRD,W(W1))) 6000,1090,1090

COMPUTE SECOND BYTE OF BARCODE ROW AND CONVERT TO BINARY
THE SECOND BYTE IS SPLIT INTO TWO PARTS. THE 4 HIGH ORDER BITS
CONTAIN THE PROGRAM TYPE (1=NONPRIVATE AND 2=PRIVATE); AND THE
4 LOW ORDER BITS CONTAIN THE SEQUENCE NUMBER, WHICH IS THE BAR-
CODE ROW NUMBER MINUS 1, MODULO 16.

1090   SECON=16+MOD(SEQNUM,16)
     SEQNUM=SEQNUM+1
     W1=11
     CHECK=CHECK+SECON
     IF(PIR,T.GT.255) CHECK=CHECK-255
     IF(PIR,T.GE.10) WRITE(6,551)ROW,SECON,CHECK
     IF(INST(SECND,W(W1))) 6000,1180,1180
COMPUTE FIRST BYTE OF BARCODE RCW 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,...).

1180   \( W1=3 \)
FIRST=CHECK
IF(IPRT,GE,10) WRITE(6,555)W,FIRST,CHECK
IF(AINS(FIRST,W(W1))) 6300,1220,1220

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

1220   IF(IPRT,GE,20) WRITE(6,556)
556   FORMAT(' END OF BARCODE ROW*******************************')
W11=0
W(2)=0
W(WP)=1
ENDING=WP+1
W(ENDING)=0
ENDBIT=WP+2
W(ENDBIT)=-99

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

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

INSERT CALL TO VERSATEC HERE.

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

IF(IPRT,GE,20) WRITE(6,558) B1,ENDBIT
558   FORMAT(' TRANSFER TO BINARY ARRAY AT',',17', 'NUMBER DIGITS', '15')
DO 1350 I=1,ENDING
    IF(W(I)) EQ,1) GOTO 559
    ALPHA(I)=ZERO
    GOTO 1350
1350   ALPHA(I)=ONE

CONTINUE IF(ENDING.EQ.132) GOTO 1736
DO 1735 I=ENDBIT,132
ALPHA(I)=BLNK
CONTINUE
WRITE(*,777) (ALPHA(I),I=1,132)
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)
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
BSTR$=0
M1=1
RETURN

ERROR HANDLING SECTION FOLLOWS
WRITE(6,6001) FUNC$***ERROR IN BARCODE PRODUCTION *** ','A4)
FMI=P-1
WRITE(6,6010) SEQNUM,PMI,M(PMI),CHECK
FORMAT(*',13,'M(',I4,') OPERAND=',I3,' CHECKSUM=',I4)
BSTR$=-1
RETURN
END
**INTEGER FUNCTION COMPS(A*,LA,M,M1)**

************************************************************************************
**COM00010**
**C**
**C**
**THIS IS THE MASTER INSTRUCTION INTERPRETATION ROUTINE FOR THE**
**C**
**HP41C COMPILER. THIS ROUTINE IS USED BY BOTH THE BAR CODE**
**C**
**GENERATOR AND THE CALCULATOR EMULATOR.**
**C**
**C**
**INSTRUCTIONS ARE PASSED TO THIS ROUTINE ONE AT A TIME IN**
**A TEXT STRING AS THE ARRAY M IS THE TOTAL ARRAY OF DECIMAL**
**INTEGER KEY CODES (MACHINE INSTRUCTIONS), AND M1 IS THE**
**POSITION WHERE THE NEXT DECODED MACHINE INSTRUCTION WILL BE**
**PLACED. Thus, the input to this routine IS A TEXTUAL HP41C**
**INSTRUCTION AND THE OUTPUT IS ONE OR MORE DECIMAL KEY CODES**
**PLACED APPROPRIATELY INTO ARRAY M.**
**C**
**C**
**THE RETURN VALUE OF THE FUNCTION COMPS IS SET AS FOLLOWS:**
**C**
**1 = END STATEMENT FOUND, END COMPILATION.**
**C**
**0 = CONTINUE TO COMPIL E**
**C**
**-1 = AN ERROR IN COMPILING THE INSTRUCTION.**
**C**

************************************************************************************
**COM00240**

**IMP LICT INTEGER(A-Z)**

**COM00250**

**COMMON/TEXT/IDIM,IPRT**

**COM00260**

**COMMON/FLAG/DONE, DIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2**

**COM00270**

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

**COM00280**

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

**COM00290**

**INTEGER# P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2**

**COM00300**

**INTEGER# A#(100)**

**COM00310**

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

**COM00320**

**INTEGER# S1$(40),S2$(40),S3$(40),S4$(40),S5$(40)**

**COM00330**

**INTEGER# L1,L2,LSAV,LWORK**

**COM00340**

**INTEGER# LS1,LS2,LS3,LS4,LS5**

**COM00350**

**INTEGER# M(1),M1**

**COM00360**

**INTEGER# FUNC#COM**

**COM00370**

**INTEGER# END#E,N,D,**

**COM00380**

**RCL(3)/R/,C,"L**,Q UOTE,"/**

**COM00390**

**APPEND/","**

**COM00400**

**LBL(3)/L,"B","D**,GTO(3)/G**

**COM00410**

**XEQ(3)/X","E","Q**,XRO(3)/X","R","D**

**COM00420**

**STO(3)/S","T","D","P","P**,IND(3)/I","N","D**

**COM00430**

**COM00440**

**COM00450**

**COM00460**

**COM00470**

**COM00480**
C 2      END2(5)'/..E',..N',..D',..'/', COM00490
C 2      AQUOTE(2)'/..A',..'A/', COM00503
C 2      MINUS,f=/', COM00506
C 2      INTEGER# 0RCL(5)/144,32,'R',.'C',.'L',.'MSTD(5)/145,48,'S',.'T',.'O'/ COM00510
C 200     IFIPRT.GE.101 WRITE(6,200)LA,AS{i},110A1 COM00520
C 200     IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000 COM00530
C C C      SET FLAGS AND INITIALIZE COUNTERS
C C C      INDIR=.FALSE.
C C C C      CHECK FOR NULL STRING ENTRY INTO COMP$
C 5      IF(LA.NE.0) GOTO 10 COM00540
C 5      COMPS=0 COM00550
C 5      RETURN
C C C C      CHECK FOR CATEGORY THREE SPECIAL INSTRUCTIONS
C 10     IF(EQS(AS,LA,RCL,3,3)) 6000,15,11 COM00560
C 11     COMPS=MEM$(AS,LA,M,M1,HRC1)
C 11     PDIRIT=.FALSE.
C 11     RETURN
C 15     IF(EQS(AS,LA,STO,3,3)) 6000,20,16 COM00570
C 16     MAKE A QUICK CHECK FOR THE STOP INSTRUCTION
C 16     IF(A$(4,EP) GOTO 65 COM00580
C 16     COMPS=MEM$(AS,LA,M,M1,MSTD)
C 16     PDIRIT=.FALSE.
C 16     RETURN
C 20     IF(EQS(AS,LA,LBL,3,3)) 6000,25,21 COM00590
C 21     COMPS=LBS$(AS,LA,M,M1)
C 21     PDIRIT=.FALSE.
C 21     RETURN
C 25     IF(EQS(AS,LA,GT0,3,3)) 6000,30,26 COM00600
C 26     COMPS=GT0$(AS,LA,M,M1)
C 26     PDIRIT=.FALSE.
C 26     RETURN
C 30     IF(EQS(AS,LA,XEQ,3,3)) 6000,35,31
31       COMPS = XEQ$(A$, LA, M, M1)
32       PDIGIT = .FALSE.
33       RETURN
34 IF(EQ$(A$, LA, XOR, 3, 3)) 6000, 40, 36
35       COMPS = XOR$(A$, LA, M, M1)
36       PDIGIT = .FALSE.
37       RETURN
38 IF(EQ$(A$, LA, END, 3, 3)) 6000, 45, 41
39       COMPS = END$(A$, LA, M, M1)
40       PDIGIT = .FALSE.
41       RETURN

CHECK FOR ALPHABETIC ENTRY INSTRUCTION.

45 IF(A$(1).NE.QUOTE) GOTO 50
46       COMPS = ALPI$(A$, LA, M, M1)
47       PDIGIT = .FALSE.
48       RETURN

CHECK FOR NUMERIC ENTRY INSTRUCTION.

50 IF(NUMC$(A$, LA, IANSW) 6000, 55.51
51       COMPS = DIGT$(A$, LA, M, M1)
52       PDIGIT = .TRUE.
53       RETURN

CHECK FOR CATEGORY ONE INSTRUCTION BY LOOKING FOR BLANK

55 PL = POS$(A$, LA, BLNK, 1, 1)
56 IF(PL) 6000, 65, 70

NC BLANK IN STRING IMPLIES HAVE FOUND ONE BYTE INSTRUCTION

65       COMPS = IONE$(A$, LA, M, M1, 1)
66       PDIGIT = .FALSE.
67       RETURN
BLANK IN STRING MEANS MULTI-WORD INSTRUCTION, NOW EXTRACT PREFIX

70 IF(PARS$(A$,LA,SS1$,LSS1$)) 6000,65,75

CHECK FOR INDIRECT ADDRESSING

75 P6=POS$(A$,LA,IND,3,1)
IF(P6) 6000,80,76

INDIR=.TRUE.,
IF(IPR.GE.20)WRITE(6,235)
FORMAT('*** DETECTED INDIRECT GOTO INSTRUCTION*')
IF(LCUT$(A$,LA,3)) 6000,6080,80

COMPILE THE PREFIX OF A MULTI-WORD INSTRUCTION

80 COMP$=IONES(SS1$,LSS1,M,M1,2)

EXTRACT THE POSTFIX OF A MULTI-WORD INSTRUCTION

85 IF(PARS$(A$,LA,SS2$,LSS2$)) 6000,90,6090

COMPILE THE POSTFIX OF A MULTI-WORD INSTRUCTION

90 COMP$=MINO(COMP$,ITWO$(SS2$,LSS2,M,M1,INDIR))
PDIGIT=.FALSE.,
RETURN

ERROR HANDLING SECTION FOLLOWS

6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
COMP$=-1
RETURN

6080 WRITE(6,6081)
6081 FORMAT('**** ERROR ')
CCMP$=-1
INTEGER FUNCTION CONS(A$, LA, LB, C$, LC)

STRING A$ AND STRING B$ ARE CONCATENATED AND PLACED IN C$.
IT IS FEASIBLE TO CONS(A$, LA, LB, A$, LA) OR
CONS(A$, LA, LB, 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 CONS, 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, IPRINT
INTEGER*2 AS(IDIM), BS(IDIM), CS(IDIM)
INTEGER*4 FUNC$/$CON$/$
IF(IPRINT.GE.10) WRITE(6,200)LA, (AS(I), I=1, LA)
IF(IPRINT.GE.10) WRITE(6,201)LB, (BS(I), I=1, LB)
200 FORMAT('TRACE $', 'CON$:', '110A1')
201 IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6030
IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000

C

LOSS=0

Determine Length of Result

ILC=LA+LB
IF(ILC.LE.IDIM) GOTO 20

LOSS=ILC-IDIM
ILB=LB-LOSS
ILC=IDIM
IF(IPRINT.GE.05) WRITE(6,202) LOSS
202 FORMAT('LOSS OF', '13,', 'CHARACTERS DURING CONCATENATION')
GOTO 25

20
ILB=LB
25
ILA=LA
INDEX=ILC

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

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

C
SET LENGTH OF C$ AND ASSIGN VALUE OF CONSC$ AND RETURN.
C
LC=ILC
60 IF(IPRT.GE.20) WRITE(6,203)LIC,(C$[I],I=1,LC)
203 FORMAT(' CONCAT: LIC=' ,I3, ' ' ,I10,A1)
IF(LOSS.NE.0)GOTO 70
CONC$=ILC
GOTO 75
70 CONC$=LOSS
GOTO 75
75 RETURN

C
ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
CONC$=-1
RETURN
END
INTEGER FUNCTION DIGIT$((AS, LA, M, M1))

**DIG00000**
**DIG00001**
**DIG00002**
**DIG00003**
**DIG00004**
**DIG00005**
**DIG00006**
**DIG00007**
**DIG00008**
**DIG00009**
**DIG00010**
**DIG00011**
**DIG00012**

C** 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.

C

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
COMMON/FLAGS/DONE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2
LOGICAL DONE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2
INTEGER# M, IDIM
INTEGER# PLUS/
INTEGER# C$(13)"0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20'
2
&
INTEGER# LC/13/
INTEGER# FUNC$"DIGIT"/
INTEGER# M1
LOGICAL POECDM, CHSFLG
 IF(IPRT.GE.10) WRITE(6,200)LA, (AS(I), I=1, LA)
200 FORAMIT"TRACE", '13', 'DIGIT$': '110A1'
 IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C ADD A NULL INSTRUCTION BETWEEN ADJACENT DIGIT ENTRY INSTR.
C
 IFI.NOT.PDIGIT) GOTO 400
C
ADJACENT DIGIT ENTRY INSTRUCTION FOUND
IBYTE=1
M(M1)=BYTE
 IF(IPRT.GE.20) WRITE(6,212)ML, IBYTE
212 FORAMIT"DIGIT", '15', "LENGTH OF THIS INSTR IS": '113'
 M1=M1+1
M(M1)=0
 IF(IPRT.GE.20) WRITE(6,213)ML, M(M1)
213 FORAMIT"DIGIT", '15', "NULL INSTR FOR PRECEDING DIGIT ENTRY", T75, '13'
2
 M1=M1+1
C
C
C CHECK FOR DIGIT ENTRY INSTRUCTION PRECEEDED BY PLUS SIGN.
400 IF(A$(1).NE.PLUS)GOTO 450
C NOTE THAT YOU GO AROUND THE FOLLOWING LINE IF THE FIRST
C DIGIT IS NOT A PLUS SIGN OR IF THE PLUS SIGN IS ALL ALONE.
C A PLUS SIGN BY ITSELF INDICATES ADDITION NOT A DIGIT
C ENTRY INSTRUCTION. ADDITION IS COMPILED BY A TABLE LOOKUP.
C CALL LCUT$(A$,LA,1)
C
C SET THE LENGTH OF THE INSTRUCTION
450 IBYTE=LA
M(M1)=IBYTE
IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215 FORMAT('DIGIT$',15,'LENGTH OF THIS INSTR IS',I3)
M1=M1+1
C
C DO 35 I=1,LA
15 IZ=I%D$(A$(I),LA,CS,LC,LOC)
IF(IZ.NE.0)GOTO 20
WRITE(6,207)M1
207 FORMAT('***** INVALID CHARACTER *****',5X,I5)
DIGTS=1
RETURN
20 M(M1)=IZ+15
IF(IPRT.GE.10)WRITE(6,208)M1,CS$(IZ),M(M1)
208 FORMAT('DIGIT$',15,'DIGIT ENTRY INSTR ','3X,A1,3X,'T75,I3)
M1=M1+1
35 CONTINUE
C
40 DIGTS=0
RETURN
C
C ERROR HANDLING SECTION OCCURS
C 6000 WRITE(6,6001)FUTC$ E''
6001 FORMAT(1*** STRING LENGTH ERROR *** ',A4)
6010 WRITE(6,6010) LA, LB, IDIM
       FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
       DIGITS=-1
       RETURN
6999 WRITE(6,602)M1
602 FORMAT('***** DIGIT ENTRY INSTR ERROR ***,5X,15)
       DIGITS=-1
       RETURN
       END
INTEGER FUNCTION END$(A$, LA, M, M1)  

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

** THE RETURN VALUE OF THE FUNCTION END$ 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, 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 /CNTE **

** 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$(IDIM) **

** INTEGER*2 BLNK/**

** INTEGER*4 FUNC$/*END$/**

** INTEGER*4 M1(1) **

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

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

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

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

M1(1)=3

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

201 FORMAT('END$ ', '13', 'LENGTH OF NEXT INSTR IS', '13')

M1=M1+1

LOAD PREFIX CODE FOR END INSTRUCTION. 193 IS USED INSTEAD OF 192. END00450

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

ENABLE 192 TO STAND FOR AN ALPHABET 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

END00490

END00500

END00510

END00520

END00530

END00540

END00550

END00560

END00570

END00580

END00590

END00600

END00610

END00620

END00630

END00640

END00650

END00660

END00670

END00680

END00690

END00700

END00710

END00720

END00730

END00740

END00750

END00760

END00770

END00780

END00790

END00800

END00810

END00820

END00830

END00840

END00850

END00860

END00870

END00880

END00890

END00900

END00910

END00920

END00930

END00940

END00953

PROVIDE TWO NULL INSTRUCTIONS TO RESERVE SPACE FOR THE LINK

POINTER. ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN

POINTER 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(6,211)M1,M(M1)

211 FORMAT('END$,15,' TRAILING NULL INSTR',T75,13)

M1=M1+1

END00690

END00700

END00710

END00720

END00730

END00740

END00750

END00760

END00770

END00780

END00790

END00800

END00810

END00820

END00830

END00840

END00850

END00860

END00870

END00880

END00890

END00900

END00910

END00920

END00930

END00940

END00953

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('ICOMPIllATION ENDED$,15,' MACHINE CODES GENERATED')

END00830

END00840

END00850

END00860

END00870

END00880

END00890

END00900

ERROR HANDLING SECTION follows

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)

END$=-1

RETURN

END00930

END00940

END00953
INTEGER FUNCTION EQ$(A$,LA,B$,LB,NUM)

**This function tests for string equality. For this function the return value is crucial as it contains the results of the test for equality.**

**Number defines the number of characters to be examined for equality, starting from the left most position of both strings. Thus, the strings:**

A$='ABCDEG' and B$='ABC'

**Will be "equal" if tested with EQ$(A$,LA,B$,LB,LIB) but "unequal" if tested with EQ$(A$,LA,B$,LB,LIB).**

**To test for absolute equality, just assign NUM to be some arbitrarily large integer, say 1000. The comparison will terminate appropriately at the end of the shortest string.**

IE. EQ$(A$,LA,B$,LB,1001) will test absolute equality

**It is suggested that this routine be used in an arithmetic statement of the form:**

IF EQ$(A$,LA,B$,LB,LIB) 6002, 10, 20

**Where: 6002 is an error handling routine 10 is the routine when strings are not equal 20 is the routine when strings are equal**

**Common/text/idim,iprt integer*2 a$(idim),b$(idim)
integer*4 func$/'eq'/

integer*4 la,lb,num

IF(IPRT.GE.10) WRITE(6,230)LA,NUM,(A$(1),I=1,LA)

200 FORMAT('TRACE ','13','EQ$(1,'13','):','110A1)

199 FORMAT(' ',13,':','110A1)

IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6060

IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6060

C

LENGTH=NUM

END
IF((LENGTH.LE.LA).AND.(LENGTH.LE.LB))GOTO 10
IF(ALA.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('COMPARE A$(',I3,'$)'=','A$I,',' WITH B$(',I3,'$)'=','A$I)
   IF(A$(I).EQ.B$(I)) GOTO 15
   EQ$=0
   IF(IPRT.GE.20)WRITE(6,202)
   FORMAT('STRINGS FOUND UNEQUAL',I3,' POSITION')
RETURN

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$       *** STRING LENGTH ERROR *** ,A4)
   FORMAT(6,6010) LA,LB,IDIM
WRITE(6,6010) LA,LB,IDIM
   FORMAT(6,6010) LA,LB,IDIM
   EQ$=9999
RETURN
END
INTEGER FUNCTION FINDS(A$, LA, B$, LB, LOC)

"FIND A$ IN B$"

STRING B$ IS SEARCHED FOR THE FIRST OCCURRENCE OF A MATCH WITH CHARACTER A$. A$ 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 FINDS IS SET TO LOC (LOCATION OF FIRST MATCH IN B$) IF MATCH FOUND

-1 IF AN ERROR IS ENCOUNTERED.

IMPPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 A$(1), B$(IDIM)
INTEGER*2 OBJECT
INTEGER*4 FUNC$/'FIND$'
INTEGER*4 LOC,FIND$200 IF(IPRT.GE.10) WRITE(6,200)LA,A$(1)
FORMAT('TRACE'*,'13,' FIND$ : ' '110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000

OBJECT=A$(1)
INDEX=1
DC 25 I=1 LB
IF(IPRT.GE.40) WRITE(6,211) OBJECT,I,B$(1)
FORMAT('COMPARE OBJECT','1A1', ' WITH B$','13,','*','A1')
IF(OBJECT.EQ.B$(INDEX)) GOTO 30
INDEX=INDEX+1
CONTINUE

NO MATCH FOUND
LOC=0
FIND$=0
IF(IPRT.GE.20) WRITE(6,201) LOC

FIN00010 FIN00020
**FIN00030 **FIN00040 **FIN00050
**FIN00060 FIN00070 FIN00080
**FIN00090 FIN00100 FIN00110
**FIN00120 FIN00130 FIN00140
**FIN00150 **FIN00160 **FIN00170
**FIN00180 FIN00190 FIN00200
FIN00210 FIN00220 FIN00230
FIN00240 FIN00250 FIN00260
FIN00270 FIN00280 FIN00290
FIN00300 FIN00310 FIN00320
FIN00330 FIN00340 FIN00350
FIN00360 FIN00370 FIN00380
FIN00390 FIN00400 FIN00410
FIN00420 FIN00430 FIN00440
FIN00450 FIN00460 FIN00470
FIN00480
FORMAT(* NO SINGLE CHARACTER MATCH FOUND,,12)
RETURN

HAVE FOUND A MATCH
LOC=INDEX
IF PRINTED IF INDEX > 30 WRITE(6,202) LOC
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(*,6001) FUNC$,
       **+**
       STRING
       ERROR
       **+**
       A4!
       IDIM=L6=,L110,
       IDIM=,L110
RETURN
END

178
INTEGER FUNCTION GTO$(A$, LA, M, M1)

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

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

IMPLIED INTEGER(LA-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
COMMON/CNTR/DONE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2
INTEGER*4 P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2
INTEGER*2 A$(IDIM)
INTEGER*2 BNLK/* */
INTEGER*2 QUOTE/* " */
INTEGER*2 IND(3)/'I','N','D'/
INTEGER*2 LABEL(26)/'A','B','C','D','E','F','G','H','I','J','

IF(IPRT,GE,10) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT('I3',' GTO$ : ',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 GOTO WITHOUT IND)

IBYTE=3
PREFIX=208

STRIP STRING OF "GTO" CHARACTERS.
CALL LCUT$(A$, LA,3)
IF(TRIM$(A$, LA)) 6015, 6015, 10

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

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

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

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

OPERAND MUST BE REGISTER X,Y,Z,T CR L OR A LOCAL ALPHA LABEL
25 DO 30 I=1,26
26 INDEX=I
27 IF(A$(I).EQ.LABEL(I)) GOTO 35
28 CONTINUE
30 CONTINUE
31 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-
35 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(MI)=IBYTE
IF (IPRT.GE.20) WRITE(6, 213) M1, IBYTE
M1=M1+1

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

M(MI)=177+INDEX
IF (IPRT.GE.10) WRITE(6, 213) M1, M(MI)
213 FORMAT(' GTO*15, 15, "TWO BYTE GTO INSTR", T75, T13)
M1=M1+1

LOAD NULL INSTR FOR TWO BYTE GTO INSTR AND RETURN

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

PROCESS THE GOTO INSTRUCTION (OPERAND>14)
75     M(M1)=1BYTE
    IF(IPRT.GE.10)WRITE(6,210)M1,BYTE
    M1=M1+1

210   FORMAT('GTO$','15,1','LENGTH OF NEXT INSTR IS','13)

LOAD THE GTO INSTR PREFIX

    M(M1)=PREFIX
    IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
    M1=M1+1

LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTER)

    IF(INDIR)GOTO 95
    M(M1)=0
    IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)
    M1=M1+1

LOAD THE 2D OPERAND OF THE GTO INSTR

    IF(INDIR)INDEX=INDEX+128
    M(M1)=INDEX
    IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
    M1=M1+1
    GTO4=0
    RETURN

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%************%************%************%************%************%

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE

80     K=0
    P2=POS$(A$2,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 LBL H=4 FOR XEQ H=2
H=2
IBYTE=H+LENGTH
M(M1)=IBYTE
IF(I$PR+GE.20)WRITE(6,210)M1,IBYTE
M1=M1+1

HAVE FOUND VALID ALPHANUMERIC LABEL, LCAD "GTO" INSTRUCTION

PREFIX=29
FOR GTO PREFIX=29 FOR LBL PREFIX=192 FOR XEQ PREFIX=30
M(M1)=PREFIX
IF(I$PR+GE.10)WRITE(6,214)M1,M(M1)
FORMAT( GTO$","5,"ALPHA GTO INSTR","T75","I3"
M1=M1+1

SET INDICATOR FOR NUMBER OF ALPHA CHAR IN LABEL

U=240
FOR GTO U=240 FOR LBL U=241 FOR XEQ U=240
M(M1)=U+LENGTH
IF(I$PR+GE.10)WRITE(6,216)|M1,M(M1)
FORMAT( GTO$","5,"LENGTH CODE ALPH GTO","T75","I3"
M1=M1+1
ADD ALPHABETIC CHARACTERS AND RETURN

140 LA=LENGTH+1
     GTOS=ALPH$(A$,LA,M,M1)
     RETURN

ERROR HANDLING SECTION Follows

6000 WRITE(6,6001) FUNC$
6001 FCMAT((** STRING LENGTH ERROR ** ',A4)
6010 WRITE(6,6010) LA, LB, IDIM
     FORMAT(' LA=','110,'LB=','110,' IDIM='110)
     GTOS=-1
     RETURN
6015 WRITE(6,6016)
6016 FCMAT((**** INVALID SECOND OPERAND IN GTO INSTR ****')
     GTOS=-1
     RETURN
6020 WRITE(6,6021)
6021 FCMAT((**** FOUND THREE OPERANS, EXPECTING IND ****')
     GTOS=-1
     RETURN
     END
INTEGER FUNCTION IN$(A$, LA, IN)

STRING A$ IS READ FROM UNIT IN.

THE LENGTH OF A$ 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
CHARS.

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

CHECK FOR END OF FILE

IF(NOT.EOFFILE(IN)) GOTO 5
IN$=-1
LA=0
IF(IPRT.GE.20) WRITE(6,201) IN
201 FORMAT(* ATTEMPT TO READ AFTER END OF FILE ON UNIT ',12)
RETURN

READ THE ACTUAL CARD

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)

CHECK CARD FOR TRAILING BLANKS

IN=0
OG 60 I=1,80
INDEX=81-1
IF(CARD(INDEX).NE.BLNK) GOTO 65
IF(IPLR,GE,20) WRITE(6,208)
208 FORMAT('FOUND INPUT STRING IS ALL BLANKS')
RETURN
70 IEND=80-IM

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

DETERMINE LENGTH OF INPUT STRING
C
LA=IEND-IBEG+1
IF(LA.LE.IDIM) GOTO 30
   LOSS=LA-IDIM
   IF(IPLR,GE,101) WRITE(6,216) LOSS
216 FORMAT('STRING TOO LONG FOR MAX STRING LENGTH, LOST')
   LA=IDIM
   IEND=IEND-LOSS
C
TRANSFER THE CARD CHARACTERS TO THE INPUT STRING.
C
INDEX=1
DC 85 I=IBEG,IEND
A$(INDEX)=CARD(I)
IF(IPRT.GE.30) WRITE(6,209) I,CARD(I),INDEX,A$(INDEX)
209 FORMAT(2 MOVE CARD('1,13,'')=*,'A1,' IS NOW C('1,13,'')='A1)
INDEX=INDEX+1
CONTINUE

CHECK FOR STRING ERROR AND RETURN

IF(IPRT.GE.20) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT(2 TRACE ',13,' IN$ = ',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
IN$=LA
RETURN

HANDLE END OF FILE CONDITION

999 ECFILE(IN$)=.TRUE.
IN$=-1
LA=0
IF(IPRT.GE.20) WRITE(6,215)
215 FORMAT(2 END OF FILE ENCOUNTERED*)
RETURN

ERROR HANDLING SECTION FOLLOWS

6000 WRITE(6,6001) FUNC$
6001 FORMAT(2 *** STRING LENGTH ERROR *** ',A4)
IN$=-1
RETURN
END
INTEGER FUNCTION IONE$(AS, LA, M, ML, IBYTE)

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

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRIL, COMMON/FLAGS/DONE, D0111, ALPH, DIG1, ALPH, IND1, INDIR, FLg1, FLAG2
COMMON/TABLE/INST, LINST, CODE, NINST
INTEGER#(6, 111)
INTEGER#(6, 111, CODE(111), NINST
INTEGER#2 A#(IDIM)
INTEGER#4 FUNC#"I ONE/
INTEGER#4 M(I)
IF(IPRT.GE.10) WRITE(6, 200) LA, (AS(I), I=1, LA)
FORMAT(* TRACR ', (3, IONE$ : '110111)
IF(LA.GT. IDIM, OR, LA.LT.0) GOTO 6000

DO 50 I = 1, NINST
LENGTH=LINST(I)
IF(LA.NE.LENGTH) GOTO 59
DO 30 J = 1, LENGTH
IF(AS(J).NE.INST(J, I)) 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(1).LE.143) GOTO 35
IF(IBYTE.EQ.2, AND, CODE(1).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$','15',' LENGTH OF NEXT INSTR IS','13')
M1=M1+1
M1=M1+CODE(1)
IF(IPRT.GE.10) WRITE(6,210)M1,(INSTR(1:1),JJ=1,6),M(M1)
210 FORMAT('IONES$','15','1,6A1','1, INSTR(1:75,13)
M1=M1+1
C IF(LA.GT.LENGTH) GOTO 35
LA=0
IONES$=0
RETURN

Following comment lines have been retained to facilitate use of
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.

35 LEFT=LA-LENGTH
LASTART=LENGTH+1
IF(SEG$($A$,A$,LASTART,LEFT),GT,0) GOTO 40
IONES$=0
RETURN
IONES$=LA
RETURN

Instruction does not match so check next instruction
Continue

No match found

60 WRITE(6,215)
215 FORMAT('**** UNRECOGNIZABLE INSTRUCTION *****')
M(M1)=1BYTE
IF(IPRT.GE.20) WRITE(6,212)M1,1BYTE
M1=M1+1
M(M1)=0
IF(IPRT.GE.10) WRITE(6,211)M1,M(M1)
211 FORMAT('IONES$','15',' NULL INSTR(1:75,13)
M1=M1+1
C IONES$=1
RETURN
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ','A4')
6010 FORMAT(' LA=',110,' LB=',110,' IDIM=',110)
    ICN=ICN-1
    RETURN
6020 WRITE(6,6021)
6021 FORMAT(' *** LENGTH OF INSTRUCTION DOES NOT MATCH NUMBER OPERNDS')
    ICN=ICN-1
    RETURN
    END
INTEGER FUNCTION IRDR$(!DEV,VAL)
C******************************************************************************
** THIS FUNCTION READS A FREE FORMAT VALUE AND CONVERTS IT TO 
** AN INTEGER.
C******************************************************************************
IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIN,IPRT
INTEGER*2 A$(*0)
INTEGER*2 BLNK/*0*/ZERO/*0*/MINUS/*-*/
DATA LL/256/
INTEGER*4 RVAL,SIGN,IFN,FRA
INTEGER*4 IRDR$,VAL,DEV
IF(IPRT,GE,20)WRITE(6,200)DEV
200 FORMAT(' TRACE IRDR$, ',I5)

IF(IN$(A$,LA,DEV))6000,12,12

12 SIGN=1
IFN=0
15 DG 1 II=1,LA
213 FORMAT(' EXAMINING ',A$ FOR INTEGER VALUE')
20 IF(A$(II),EQ.MINUS)GO TO 10
IF(A$(II),EQ.BLKNK)GOTO 1
IF(IPRT,GE,30) WRITE(6,213)A$(II)
10 IFN=IFN+10+ITEMP
GO TO 1
10 SIGN=-1
214 FORMAT(' FOUND MINUS SIGN')
20 CONTINUE
217 FORMAT(' SUBROUTINE IRDR$ RETURNING VALUE ',I20)
2225 FORMAT(' SET FINAL SIGN OF ',I20, ' WITH ',I20)
660 IRDR$=SIGN*IFN
CROSS COMPILER
AND
PROGRAMMING SUPPORT SYSTEM FOR THE HP41CV --ETC(u)
SEP 81  J N RICHMANN
VAL=IRDS$  
IF(IPRT.GE.20)WRITE(6,217)VAL  
RETURN

C  
C ERROR HANDLING SECTION Follows

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

6007 WRITE(6,6008)(A$(I),I=1,LA)
6008 FORMAT('*** ATTEMPT TO FIND INTG VALUE OF ALPHABETIC STRING:','110A1')  
STOP
END
INTEGER FUNCTION ITWO$(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 COMPILCE
-1 = AN ERROR IN COMPILING THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRI
INTEGER*2 A$(IDIM),
INTEGER*2 BLNK/

INTEGER*2 LABEL(26) /'A$'. 'B$'. 'C$'. 'D$'. 'E$'. 'F$'. 'G$'. 'H$'. 'I$'. 'J$'.

INTEGER*4 FUNC$/'ITWO$'/
INTEGER*4 M(1)
LOGICAL INDIR
IF(IPRI.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)

200 FORMAT('TRACE ',I3,' ITWO$: ',I10A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

CHECK FOR BLANK INDICATING A BUGUS THIRD OPERAND

IF( TRIM$(A$,LA)) 6015,6030,10
IF( POS$(A$,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=1,126
   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 OPER-
AND, THEN GOTO PROCESS SECTION TO ACTUALLY LOAD BYTE.

INDEX=INDEX+101
GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

IF(IVAL$(A$,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(1)=INDEX
IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212   FORMAT(' ITWO$','1S',' 2D OPERAND ','T75,13'
       M1=M1+1

CLEAN-UP AND RETURN

INDIR=.FALSE.
ITWO$=0
RETURN

ERROR HANDLING SECTION Follows

6000 WRITE(6,6001) FUNC$
6001 FORMAT(*** STRING LENGTH ERROR *** ','A4)
6010 WRITE(6,6010) LA,LB,IDIM
6015 RETURN

6016 WRITE(6,6016)
6017 FORMAT('***** INVALID SECOND OPERAND IN INSTR *****')
6020 RETURN

6021 WRITE(6,6021)
6022 FORMAT('***** FOUND THREE OPERANDS, EXPECTING IND *****')
6030 RETURN

6031 WRITE(6,6031)
6032 FORMAT('***** FOUND SECOND OPERAND BLANK ******')
6035 RETURN
END
INTEGER FUNCTION IVAL$(A*$,LA*,VAL)  
***IVal(N)***  
***CONVERTS A NUMERIC TEXT STRING TO INTEGER NUMERIC VALUE.***  
***IVal(K)***  
**IN**  
**COMMON**  
**TEXT/IDIM,IPRT**  
**INTEGER*2 AS{DIM}**  
**INTEGER*2 BLNK'/ '/ZERO/*O*/MINUS/*-*/**  
**INTEGER*4 FUNCT/'VAL/*  
**DATA LL/256/**  
**INTEGER*4 RVAL,SIGN,IFN,FRAC**  
**INTEGER*4 IVAL$,VAL**  
IF(IPRT.GE.10) WRITE(6,200)LA$,A$(I),I=1,LA$  
200 FORMAT('TRACE',',13,',',IVAl$(,',',110A1)')  
IF(LA$.GT.IDIM.OR.LA$.LT.0) GOTO 6000  
  
C  
SIGN=1  
IFN=0  
15 DC I=1,LA$  
213 FORMAT('EXAMINING ',A$, ' FOR INTEGER VALUE')  
IF(A$(I).EQ.MINUS) GO TO 10  
IF(A$(I).EQ.BLNK) GOTO 1  
20 ITEMP=A$(I)-ZERO  
ITEMP=ITEMP/LL  
IF(ITEMP.GT.9).OR.(ITEMP.LT.0) GOTO 6007  
IF(IPRT.GE.30) WRITE(6,216) ITEMP  
216 FORMAT('FOUND NUMERIC DIGIT',',I2)  
IFN=IFN+10+ITEMP  
GO TO 1  
10 SIGN=-1  
IF(IPRT.GE.30) WRITE(6,214)  
214 FORMAT('FOUND MINUS SIGN')  
CONTINUE  
217 FORMAT('SUBROUTINE IVAL$ RETURNING VALUE ',',I20)  
IF(IPRT.GE.30) WRITE(6,2225) IFN,SIGN  
2225 FORMAT('SET FINAL SIGN OF ',',I20, ' WITH ',',I20)
660 IVALS=SIGN*IFN
    VAL=IVALS
    IF(IPRT.GE.20)WRITE(6,217) VAL
    RETURN

C
C  ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(1*** STRING LENGTH ERROR *** ','A4)
    IVALS=-1
    STOP
6007 WRITE(6,6008)(A$(I),I=1,LA)
6008 FORMAT(1*** ATTEMPT TO FIND REAL VALUE OF ALPHABETIC STRING:*/
    STOP
    END
INTEGER FUNCTION LBS(A$,LA,M1)  LBL00010
**LBL00020**
**LBL00030**
**LBL00040**
**LBL00050**
**LBL00060**
**LBL00070**
**LBL00080**
**LBL00090**
**LBL00100**
**LBL00110**
**LBL00120**
**LBL00130**
**LBL00140**
**LBL00150**
**LBL00160**
**LBL00170**
**LBL00180**
**LBL00190**
**LBL00200**
**LBL00210**
**LBL00220**
**LBL00230**
**LBL00240**
**LBL00250**
**LBL00260**
**LBL00270**
**LBL00280**
**LBL00290**
**LBL00300**
**LBL00310**
**LBL00320**
**LBL00330**
**LBL00340**
**LBL00350**
**LBL00360**
**LBL00370**
**LBL00380**
**LBL00390**
**LBL00400**
**LBL00410**
**LBL00420**
**LBL00430**
**LBL00440**
**LBL00450**
**LBL00460**
**LBL00470**
**LBL00480**

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

THE RETURN VALUE OF THE FUNCTION LBS IS SET AS FOLLOWS:
0 = CONTINUE TO COMPILE
-1 = AN ERROR IN Compiling THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
COMMON/DIGIT/DONE, P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
LOGICAL DONE,P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTEGER P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTEGER A$[IDIM]
INTEGER S5,S1(20)
INTEGER L6,13
INTEGER BLK,/*
INTEGER QUOTE/*

2 INTEGER LABEL(26) A$,B$,C$,D$,E$,F$,G$,H$,I$, J$
3 INTEGER FUNC'/LBS'/
INTEGER M(1)
IF(IPRT;'GE.10') WRITE(6,200)LA, {A$(1),I=1,LA}
200 FORMAT('TRACER',L13,'LBL$ 2',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

STRIPE STRING OF "LBL" CHARACTERS.
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

PROCESS LOCAL LABELS, FIRST CHECK FOR NUMERIC OR SINGLE CHAR ALPHABET OR:

20 IF(LA-2) 25,50,6015
25 LENGTH OPERAND IMPLIES SINGLE CHARACTER ALPHA LOCAL LABEL
30 DO 30 I=1,26
    INDEX=I
    IF(A$1).EQ. LABEL(I)) GOTC 35
35 CONTINUE
C
C WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND
30 GOTO 6015
C
C HAVE FOUND A MATCH IN LOCAL LABEL TABLE; SET VALUE OF SECOND OPERAND
C AND THEN GOTO PROCESS A TWO BYTE INSTRUCTION.
35 INDEX=INDEX+101
C
C PROCESS A TWO BYTE INSTRUCTION
40 BYTES=2
M(M) = BYTE
45 IF(IPR1.GE.20) WRITE(6,210) M1, BYTE
210 FORMAT(' LBL$15,' LENGTH OF NEXT INSTR IS',I3)
M1=M1+1
C
C LOAD TWO BYTE LBL INSTR (EITHER SINGLE CHAR ALPHA OR 2 DIGIT NUM)
50 IF(IPR1.GE.10) WRITE(6,211) M1, M(M1)
211 FORMAT(' LBL$15,' TWO BYTE LBL INSTR IS',T75,I3)
M1=M1+1
M(M) = INDEX
55 IF(IPR1.GE.10) WRITE(6,212) M1, M(M1)
212 FORMAT(' LBL$15,' LBL 2D OPERAND IS',T75,I3)
M1=M1+1
LBL$=0
RETURN
LENGTH OF OPERAND IMPLIES MUST BE A NUMERIC LOCAL LABEL

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

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

55 IF(INDEX.GT.14) GOTO 40

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

BYTE=1
M[M]=IBYTE
IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
M1=M1+1

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

M[M]=1+INDEX
IF(IPRT.GE.10)WRITE(6,213)M1,M(M)
FORMAT('LBL',15,'ONE BYTE LBL INSTR',T75,13)
M1=M1+1
LBL$=0
RETURN

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE

80 K=0
P2=POS$(A$,LA,QUOTE,1,2)
IF(P2) 6015,120,85

LOOK FOR ANOTHER QUOTE

85 PL=P2+1
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
100      LA=LA-1
         GOTO 120
105      PSTART=P2+1
         CALL SEG$(A$,LA,SS$,LSS$,PSTART,LEFT)
         K=IVAL$(SS$,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
210      M(M1)=192
         IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
         FORMAT(' LBL$','15','ALPHA LBL INSTR',T75,13)
         M1=M1+1

PROVIDE ONE NULL INSTRUCTION TO RESERVE SPACE FOR THE LINK
PCINTER. ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
POINTER WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN. THIS
CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
BOUNDARIES WITHING 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(1) = 0
IF(IPRT,GE,10) WRITE(6,215) M(1), M(1)
215 FORMAT(' ', LBL$, 15, ' TRAILNG NULL INSTR ', T75, 13)
M1 = M1 + 1

SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL

M(1) = 241 + LENGTH
IF(IPRT,GE,10) WRITE(6,216) M(1), M(1)
216 FORMAT(' ', LBL$, 15, ' LENGTH CODE ALPH LBL ', T75, 13)
M1 = M1 + 1

ENCODE KEY ASSIGNMENT

IF(K,NE,0) GOTO 130
SINCE K=0 IMPLIES NULL KEY ASSIGNMENT
M(1) = 0
IF(IPRT,GE,10) WRITE(6,217) M(1), M(1)
217 FORMAT(' ', LBL$, 15, ' NULL KEY ASSIGNMENT ', T75, 13)
M1 = M1 + 1
GOTO 140
SINCE K=0 IMPLIES A KEY ASSIGNMENT TO BE MADE
K1 = 0
IF(K,GT,0) GOTO 135
K1 = 8
K = IABS(K)
135 A1 = K/10
B1 = MOD(K,10)
K = 16*B1+1+A1+K1
M(1) = K
IF(IPRT,GE,10) WRITE(6,218) M(1), M(1)
218 FORMAT(' ', LBL$, 15, ' LBL KEY ASSIGNMENT ', T75, 13)
M1 = M1 + 1

ADD ALPHABETIC CHARACTERS AND RETURN

LA = LENGTH + 1
LBL$ = ALPH$(A$, LA, M, M1)
RETURN
INTEGER FUNCTION MEM$(A$, LA, M, M1, WHICH)

** STRING A$ CONTAINS AN MEMORY INSTRUCTION, EITHER AN STO OR A RCL. **

** THE RETURN VALUE OF THE FUNCTION MEM$ IS SET AS FOLLOWS: **

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

** IMPLICIT INTEGER(A$-2) **

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
LOGICAL DANCE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2
INTEGER*4 P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/
INTEGER*2 IND(3)/1, 'N', 'O'/
INTEGER*2 C$(5)/'T', 'Z', 'Y', 'X', 'L'/
INTEGER*4 LC/ 5/
INTEGER*4 WHICH(5)
INTEGER*4 FUNS$/'MEM$'/
INTEGER*4 M(1)
IF(IPRT.GE.10) WRITE(6, 200) LA, (A$(I), I=1, LA)

200IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

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

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

CHECK FOR INDIRECT ADDRESS
C 20 P6=POS$(A$,LA,IND,3,1)
    IF(P6) 6000,40,21
    INDIR=.TRUE.
    IF(LCUT$(A$,LA,31) 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=FIN$(A$,1,LA,C$,LC,LC)$) 6015,6015,40
    IF(LZ.NE.0) GOTO 46
    WRITE(6,207)A$(1)
    FORMAT(14,6,14,9,5X,A1)
    MEM$=1
    RETURN
    POSTFX=IZ+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 OPRANDS
    I$=1
    PREFIX=POSTFX+H
    GOTO 100
C PROCESS TWO BYTE NUMERIC OPERANDS

CONTINUE

SET THE LENGTH OF THE INSTRUCTION

M(M1)=1BYTE
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

M(M1)=PREFIX
IF(IPRT.GE.10)WRITE(6,211)M1,(WHICH(I),I=3,5),M(M1)
211 FORMAT('MEM$,15,'3A1,'INSTR',775,13)
M1=M1+1

ENCODE THE POSTFIX OF THIS INSTRUCTION

IF(1BYTE.EQ.11)GOTO 125
IF(INDIR)POSTFX=POSTFX+128
M(M1)=POSTFX
IF(IPRT.GE.10)WRITE(6,221)M1,(WHICH(I),I=3,5),M(M1)
221 FORMAT('MEM$,15,'3A1,'INSTR POSTFIX',775,13)
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,IDIM
6010 FORMAT('LA=',110,,'LB=',110,,'IDIM=',110)

INTEGER FUNCTION NEWPGS(LINCNT, NUMPGE, TITLE$, LTITLE, MTITLE)

THIS SUBROUTINE IS USED TO PLACE MAKE THE OUTPUT LISTING AS ATTRACTIVE AS POSSIBLE. IT PERFORMS ALL THE HOUSEKEEPING FUNCTIONS ASSOCIATED WITH GOING TO A NEW PAGE.

IT SHOULD BE CALLED BY THE USE OF A STATEMENT SUCH AS

IF(MOD(LINCNT, PAGE).EQ.0) THEN
   CALL NEWPGS(LINCNT, PAGENUMBER, TITLE$, LTITLE, MTITLE)
ENDIF

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/ODIM, IPRT
INTEGER*2 TITLE$(LITLE, MTITLE)
INTEGER*2 BLNK/10/
INTEGER*2 CARD(80)
INTEGER*4 FUNC$/'NEWPG'/
297 FORMAT(/'****' )
299 FORMAT(*11, 116A1)
301 FORMAT(*14, 132A1)

PRINT THE OUTPUT PAGE HEADING
WRITE(6,299)
   IF(MITLE.LE.0) GOTO 75
   WRITE(6,200)(TITLE$(JJ,JJ), JJ=1,LITLE)
   IF(MITLE.EQ.1) GOTO 75
   DC 50 IJ = 2*MITLE
   WRITE(6,201)(TITLE$(IJ,JJ), JJ=1,LITLE)
CONTINUE

UPDATE THE PAGE COUNTER AND RESET THE LINE COUNTER
NUMPGE = NUMPGE+1
LINCNT = MITLE
WRITE(6,297)

C
C
C
EXIT
C
NEWPG$=NUMPGE
RETURN
C
C
ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$  
6001 FORMAT(1,30A1) PAGE OUTPUT  ERROR 3,4)  
NEWPGS=-1
RETURN
END
INTEGER FUNCTION NUMCS(A$ , LA , IANSW)

**FUNCTION** CHECKS TO SEE IF A STRING IS ALL NUMERIC.

**NUMERIC** IS DEFINED TO MEAN ALL DIGITS, "+" , "-" , "." , "," , "E" , OR BLNK

**RETURNED VALUE** OF THE INSTRUCTION IS 0 IF NOT NUMERIC

1 IF NUMERIC
-1 IF ERROR ENCOUNTERED

**IMPLICIT** INTEGER(A-Z)

COMMON/TEXT/IDIM , IPRT

INTEGER*2 A$(IDIM)

INTEGER*4 FUNC$('/NUMCS'/

INTEGER*4 DIGIT(15)/'0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9' ,

2)

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

200 FORMAT('TRACE ' , ' I3' , ' NUMCS ' , ' I3' , ' I0 A')

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

DC 10 I=1 , LA

IF(LA.NE.1) GOTO 3

NOT ALLOWED TO EXAMINE LAST 5 CHAR S IF LA ISONE.

LOOK = 10

GOTO 4

LOOK = 15

DO 5 J=1 , LOOK

IF(A$(I) .EQ. DIGIT(J)) GOTO 10

CONTINUE

HAVE FOUND A NON-NUMERIC CHARACTER

IANSW = 0

NUMCS = 0

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

201 FORMAT('STRING DETERMINED TO BE NON-NUMERIC ')

RETURN

CHARACTER FOUND TO BE NUMERIC , TAKE NEXT CHARACTER

CONTINUE

IF YOU GET TO HERE THE STRING MUST BE ALL NUMERIC.
IANSW=1
NUMC$=1
IF (IPRT .GE. 20) WRITE(6,202)
FORMAT('STRING FOUND TO BE NUMERIC')
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
FORMAT('STRING LENGTH ERROR', A4)
WRITE(6,6010) LA, LB, IDIM
FORMAT('LA=', 'i10', ' LB=', 'i10', ' IDIM=', 'i10')
NUMC$=-1
RETURN
END
INTEGER FUNCTION PARS$(A$, $L A$, $B$, $L B$)

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
POS$.
AN ATTEMPT TO PARSE A NULL STRING WILL RESULT IN A SPECIAL
CHARACTER BEING PLACED IN THE 1ST POSITION OF A$. A SUBSEQUENT
ATTEMPT TO REPARSE THIS STRING WILL RESULT IN A FATAL ERROR.
THIS FEATURE IS INTENDED TO PREVENT UNINTENTIONAL LOOPING OF THE
PARSE FUNCTION ON A NULL STRING. 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${^2}$,IDIM, $B$ (IDIM)
INTEGER$^2$ BLNK/$^/$$^/$
INTEGER$^4$ HALT/$^/$$^/$$^/$$^/$
INTEGER$^4$ FUNC$^/^$PARS$/^/$
IF(IPRT$^./^./^./^/.GE.,$204$) WRITE(6,200)$LA$, (A$(1),i)=1,LA$

   200 FORMAT(TRACE',5,13$^/^$PARS$$^/^$$'=4$110A1$
   IF(LA$.GT.$IDIM.OR.$LA$.LT.$0I GOTO 6000

   5   CHECK TO SEE IF INPUT STRING IS NULL
   IF(LA$.NE.$0I GOTO 5

   204 IF(IPRT$^./^./^./^/.GE.,$204$) WRITE(6,204)
      FORMAT(4$^/^$ATTEMPTED TO PARSE A NULL STRING.'$
      IF(A$(1).EQ.$HALT) GOTO 6005
      AS$(1)$=HALT
      PARS$$^/^$$=0
      LB=0
      RETURN

   205 TRIM THE INPUT STRING OF LEADING BLANKS
5 IM=0
DO 10 I=1,LA
IF(A$.EQ.BLANK) GOTO 15
IM=IM+1
10 CONTINUE
15 IF(IM.EQ.0) GOTO 25
IF(IPRT.GE.20) WRITE(6,201) IM
201 FORMAT(' FOUND',IM,' LEADING BLANKS IN INPUT STRING')
IF(IM.LT.LA) GOTO 20
LA=0
LB=0
PAR$=0
IF(IPRT.GE.20) WRITE(6,202)
202 FORMAT(' FOUND INPUT STRING IS ALL BLANKS')
RETURN
20 LEFT=LA-IM
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$.EQ.BLANK) GOTO 35
LB=LB+1
30 CONTINUE
CONSTRUCT TOKEN
35 DO 45 I=1,LB
B$(I)=A$(I)
45 CONTINUE
IF(IPRT.GE.20) WRITE(6,205) LB,(B$(I),I=1,LIB)
205 FORMAT(' PARS$ FOUND TOKEN',IM,' LB',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
A$(I)=A$(I+LB+1)
CONTINUE
LA=LEFT
CHECK $A$ FOR TRAILING BLANKS
IF(LA.EQ.0) GOTO 75
K=0
DO 60 I=1,LA
INDEX=LA-I+1
IF(A$(INDEX).NE.BLANK) GOTO 65
IM=IM+1
CONTINUE
IF(IM.EQ.0) GOTO 75
LPRINT=20
IF(IPRT.GE.20) WRITE(6,207) IM
IF(IPRT.GE.20) WRITE(6,208)
FORMAT('FIND REMAINING STRING IS ALL BLANKS')
RETURN
LA=IM
PARS$=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)
PARS$=-1
RETURN
WRITE(6,6006)
FORMAT('*** FATAL ERROR: ATTEMPTED TO PARSE A NULL STRING TWICE')
STOP
END
INTEGER FUNCTION POS$(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
ASSIGNED TO POS$. THE SEARCH BEGINS AT LSTART.
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*2 FUNC$, POS$

IF(IPRT.GE.10) WRITE(6, 200) LA, (A$(I), I=1, LA)
200 FORMAT('TRACE ', I3, ' POS$ ' I, '10A1')
IF(LA.GT.IDIM OR LA.LT.0) 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-LB+1
DO 25 I=LAST, LEFT
      JN=I-J-1
      IF(IPRT.GE.20) WRITE(6, 202) IN, A$(I), JN, B$(J)
202 FORMAT('COMPARE A$(', I3, ')'='", A$", WITH B$(', I3, ')'='", A$")'
      IF(A$(J).EQ.B$(J)) GOTO 25
      CONTINUE
      CONTINUE
POS$=I
203 FORMAT('AT POSITION', I3, ' SECOND STRING FOUND IN FIRST')
RETURN

C

5 CONTINUE
ROUTINE WILL FALL THROUGH TO FOLLOWING IF NO MATCH FOUND.
POS$=0
IF(IPRT.GE.20) WRITE(6, 204)
204 FORMAT('')
204 FORMAT(' FIRST STRING SEARCHED AND SECOND STRING NOT FOUND')
     RETURN
     ERROR HANDLING SECTION FOLLOWS
     WRITE(6,6001) FUNC$
     6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
     POS$=-1
     RETURN
     END
INTEGER FUNCTION RCUT$(A$,LA,NUM)

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

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/DIM,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$("TRACE ",13,RCUT$ : ",110A1)$
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

C C
C C C

LEFT$=LA$-NUM
IF(LEFT$) GOTO 20
IF(IPRT.GE.10)WRITE(6,202)
202 FFORMAT$("STRING REDUCED TO NULL STRING BY RCUT$:"
LA$=0
RCUT$=0
RETURN

C C

CONTINUE
LA$=LEFT$
IF(IPRT.GE.20)WRITE(6,201)LA$,(A$(I),I=1,LA$
201 FFORMAT$("STRING NOW ",14,",",110A1)$
RCUT$=LA$
RETURN

C C
ERROR HANDLING SECTION FOLLOWS

6000 WRITE$(6,6001)$ FUNC$
6001 FFORMAT$("### STRING LENGTH ERROR ",A4)$
RCUT$=-1
RETURN
END
INTEGER FUNCTION SEG$(A$, LA, B$, LB, LSTART, LOUT)

**A SUBSTRING IS EXTRACTED FROM A$ AND ASSIGNED TO B$. THE**
**SUBSTRING IN A$ BEGINS AT POSITION LSTART 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-
**ACTERS.**

**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/
INTEGER*4 FUNC$/'SEG'/
LOGICAL FIRST
200 FORMAT('13,13,' SEG$: '1,10A1')
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(' NULL 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 (1,13,1) GREATER THAN LENGTH FIRST STRING')

2 /' NEW STRING HAS ',I3,' BLANKS')
SEG$=0
RETURN
C
C
15 DO 35 I=1,LOUT
   IM=I
   IF(I.GT.(LA-LSTART+1))GOTO 25
   GOTO 35
25   B$[I]=BLNK
   IF(.NOT.FIRST) GOTO 35
   FIRST=.FALSE.
   SEG$=IM-1
   CONTINUE
   IF(FIRST) SEG$=IM
   IF(IPRT.GE.20) WRITE(6,203) SEG$
203  FORMAT(' SEG$ OBTAINED ',I3,' CHARACTERS FROM FIRST STRING')
   LB=LOUT
   RETURN
C
C
ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(*** STRING LENGTH ERROR *** ',A4)
   SEG$=-1
   RETURN
END
INTEGER FUNCTION TRIMS(A$LA)
C*********************************************************************************************
C** THIS FUNCTION STRIPS A STRING OF LEADING AND TRAILING BLANKS. **
C** THE RETURN VALUE OF THE FUNCTION IS SET TO THE LENGTH OF THE
C** STRING REMAINING AFTER THE BLANKS ARE REMOVED. **
C*********************************************************************************************
IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/
INTEGER*4 Funcs/"TRIM/"
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT(" TRAILING BLANKS IN STRING"); IPRT,GE.20) WRITE(6,208) IM
FORMAT(" FOUND STRING IS ALL BLANKS")
RETURN
70 IEND=LA-IM
C C
C CHECK A$ FOR TRAILING BLANKS
C
58 IM=)
DC 60 I=1,LA
INDEX=LA-I+1
IF(A$(INDEX).NE.BLNK) GOTO 65
IM=IM+1
60 CONTINUE
65 IF(IM.EQ.0) GOTO 70
IF(IPRT.GE.20) WRITE(6,207) IM
207 FORMAT(" FOUND 13; TRAILING BLANKS IN STRING")
IF(IM.NE.LA) GOTO 70
LA=0
TRIMS=0
IF(IPRT.GE.20) WRITE(6,208)
208 FORMAT(" FOUND STRING IS ALL BLANKS")
RETURN
30 RETURN
DO 10 I=1,IEND
IF(A$(I).NE.BLNK) GOTO 15
10 CONTINUE
IM = IM + 1

15 IF (IM.EQ.0) GOTO 25

20

IF (IPRT.GE.20) WRITE(6,211) IM

211 FORMAT(* FOUND',I3,' LEADING BLANKS IN STRING*)

25 IBEG = I + IM

DETERMINE LENGTH OF INPUT STRING

LA = IEND - IBEG + 1

IF (LA.LE.IDIM) GOTO 30

LOSS = LA - IDIM

216 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)

85 INDEX = INDEX + 1

CONTINUE

CHECK FOR STRING ERROR AND RETURN

IF (IPRT.GE.10) WRITE(6,222) LA, (A$(I), I = 1, LA)

222 FORMAT(* TRIM$ ',I3,' AFTER ','I1')A$)

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,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 COMPIL
-1 = AN ERROR IN COMPILING THE INSTRUCTION.

*** IMPLICIT INTEGER(A-Z) ***
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*4 A$(IDIM)
C
INTEGER*2 BLNK/' '/
INTEGER*2 QUOTE/' '/
INTEGER*2 IND(3)/'1','2','3'/
INTEGER*2 LABEL(26)/'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'/

INTEGER*4 FUNC$/'XEQ'/'
INTEGER*4 M11
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I)),I=1,LA)
200 FORMAT('TRACING',' ',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)

IRATE=3
PREFIX=224

STRIP STRING OF "XEQ" CHARACTERS.

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

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 LCU$(A$,LA,P1)
IF(IPRT.GE.20) WRITE(6,235)
235 FORMAT('DETECTED INDIRECT XEQ INSTRUCTION')
INDIR=TRUE.
BYTE=2
PREFIX=174

CHECK FOR NUMERIC OPERAND

IF(NUMC$(A$,LA,IANSW)) 6015,25,50

OPERAND MUST BE REGISTER X,Y,Z,T CR L OR A LOCAL ALPHA LABEL

DO 30 I=1,26
   INDEX=I
   IF(A$(I).EQ.LABEL(I)) GOTO 35
30 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
AND THEN GOTO PROCESS A THREE BYTE INSTRUCTION.
C
35 INDEX=INDEX+101
GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

50 IF(IVAL$(A$,LA,INDEX))6015,75,75

PROCESS THE GOTO INSTRUCTION (OPERAND>14)

75 M(M1)=IBYTE
IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
210 FORMAT(' XEQ$',15,' LENGTH OF NEXT INSTR IS',13)
M1=M1+1

LOAD THE XEQ INSTR PREFIX

C
211 M(M1)=PREFIX
IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
FORMAT(' XEQ$',15,' XEQ PREFIX INSTR',75,13)
M1=M1+1

LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTER)

C
221 IF(INDIR) GOTO 95
M(M1)=0
IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)
FORMAT(' XEQ$',15,' NULL FOR XEQ INSTR',75,13)
M1=M1+1

LOAD THE 2D OPERAND OF THE XEQ INSTR

95 IF(INDIR) INDEX=INDEX+128
NOTE THAT FOR XEQ IND THE HIGH ORDER BIT IS SET
M(M1)=INDEX
IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212 FORMAT(' XEQS',15,' XEQ 2D OPERAND ',T75,13) XEQ01450
M1=M1+1
XEQS=0
RETURN

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE

80 K=0
PZ=P0$$([A$1,LA,QUOTE,1,2]) IF(PZ) 6015,120,85

CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS

85 LEFT=LA-PZ
IF(LEFT) 6015,10),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
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, LCAD "XEQ" INSTRUCTION

C PREFIX=30 FOR GTO PREFIX=29 FOR LBL PREFIX=192 FOR XEQ PREFIX=30 XEQ01910
M(M1)=PREFIX
IF(IPRT,GE,10)WRITE(6,214)M1,M(M1)
214 FORMAT(' XEQ$','15,' ALPHA ' XEQ INSTR','T75,'I3)
M1=M1+1

SET INDICATOR FOR NUMBER OF ALPHA CHARs 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,'I3)
M1=M1+1

ADD ALPHABETIC CHARACTERS AND RETURN
C 140 LA=LENGTH+1
XEQ$=ALPH$(A$,LA,M,M1)
RETURN
C ERROR HANDLING SECTION FOLLOWS
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ','A4)
6010 WRITE(6,6010) LA,LB,IDIM
6010 FORMAT(' LA=',I10,,' LB=',I10,,' IDIM=',I10)
XEQS=-1
RETURN
6015 WRITE(6,6016)
6016 FORMAT(' *** INVALID SECOND OPERAND IN XEQ INSTR *****
XEQS=-1
RETURN
6020 WRITE(6,6021)
6021 FORMAT(' *** FOUND THREE OPERANDS, EXPECTING IND *****
XEQS=-1
RETURN
END
INTEGER FUNCTION XRO$(A$, LA, M, ML)  

**STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN XROM INSTRUCTION.**  
**THE XROM INSTRUCTIONS ARE SUBROUTINE CALLS TO H.P. SUPPLIED**  
**ROM ENCODED SUBROUTINES. THE SECOND AND THIRD OPERANDS MUST**  
**BE NUMERIC.**  

**THE RETURN VALUE OF THE FUNCTION XRO$ IS SET AS FOLLOWS:**  
**0 = CONTINUE TO COMPIL**  
**-1 = AN ERROR IN COMPILING THE INSTRUCTION.**  

IMPLICIT INTEGER(A-Z)  
COMMON/TEXT/IDIM, [PRT  
INTEGER*2 A$(IDIM)  
INTEGER*4 COMMA*/, */; SS1$(40)  
INTEGER*4 FUNC$*/XRO*/  
INTEGER*4 M(1)  
REAL*4 RFrac, RDF  
IF(I$PRT, GE, 10) WRITE(6, 200)LA, (A$(I), I=1, LA)  
200 FORMAT(I$ TRAC$  
IF(LA, GT, IDIM, OR, LA, LT, 0) GOTO 6000  

IF(LCUT$(A$, LA, 4)) 6015, 6020, 7  
IF(TRIM$(A$, LA)) 6015, 6020, 10  
P1=POS$(A$, LA, COMMA, 1, 1)  
P2=PI-1  
IF(SEC$(A$, LA, SS1$, LSS1$, 1, P2)) 6015, 6020, 12  
IF(LCUT$(A$, LA, PI)) 6015, 6020, 15  
IF(I$VAL$(A$, SS1$, ROM)) 6030, 20, 20  
IF(I$VAL$(A$, LA, PGM)) 6030, 25, 25  
IF(RFrac=ROM$4)  
IF(RFrac=FLOAT(ROM)/4.0)  
RDF=RFRAC-FLOAT(1$FRAC)  
RDF=256.0*RDF  
IDF=INT(RDF)  
I$SECOND=PGM+IDF
SET THE LENGTH OF THE INSTRUCTION

C 100  IBYTE=2
    M(M1)=IBYTE
    IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
    FORMAT(1,XROS$,I5,1,LENGTH OF THIS INSTR IS',I3)
    M1=M1+1

C ENCODE THE PREFIX OF THIS INSTRUCTION

C 211  M(M1)=IFIRST
    IF(IPRT.GE.10)WRITE(6,211)M1,IFIRST,ROM,M(M1)
    FORMAT(1,XROS$,I5,1,PREFIX=',I3,1,ROM=',I3,T75,I3)
    M1=M1+1

C ENCODE THE POSTFIX OF THIS INSTRUCTION

C 221  M(M1)=ISECOND
    IF(IPRT.GE.10)WRITE(6,221)M1,ISECOND,PGM,M(M1)
    FORMAT(I,XROS$,I5,1,POSTFIX=',I3,1,PGM=',I3,T75,I3)
    M1=M1+1

C C C

C 125  XROS$=0
    RETURN

C ERROR HANDLING SECTIONaultS

C 6300  WRITE(6,6001)FUNC$
C 6301  FORMAT(1,STRING LENGTH ERROR *** ',A4)
C 6010  FORMAT(I6,LA='',I10,'
XROS$=-1
    IDIM='',I10)
    RETURN
C 6015  WRITE(6,6016)
C 6016  FORMAT(1,INVALID ROM NUMBER IN XRO INSTR ****)
    XROS$=-1
    RETURN
C 6020  WRITE(6,6021)
6021 FORMAT('**** INVALID PROGRAM NUMBER IN XRC INSTR ****')
6030 RETURN
6031 FORMAT('**** NUMERIC CONVERSION ERROR IN XRC INSTR ****')
RETURN
END
C** THIS IS THE PROGRAM WHICH INTERFACES THE HP41CV CROSS COMPILES VER00010
C** TO THE VERSATEC HIGH RESOLUTION PLOTTER. IF THE CROSS VER00020
C** COMPILER IS USED ON A DIFFERENT PLOTTER, ONLY THIS ROUTINE VER00030
C** NEEDS TO BE CHANGED.
C**
C********************************************************************VER00040
REAL*4 HEIGHT,UNIT,DOLLARPLUS,CSIZE,CSPACE,TSPACE
REAL*4 PLONG,PWIDE,PFACT,ZERO,TIMAR,SMAR
INTEGER*2 IN(133),BLANK,'/',ZERO,'O','/',ONE,'I','/',TITLE(80)
IN.TEGER*4 PERPG,ROW

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-PACKARDS 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 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.
C

CSIZE=0.08*HEIGHT
TSPACE=0.22*HEIGHT

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

TSPACE=CSIZE+(1.5*CSIZE)

HCFACTR IS THE RELATIVE HEIGHT OF THE SUM OF THE BAR HEIGHT
PLUS THE SPACE BETWEEN THE BARS, INCLUDING LABELS

HCFACTR=1.7*HEIGHT

PLONG IS THE HEIGHT OF EACH PAGE OF BARCODE

PLONG=11.0

PWIDE IS THE WIDTH OF EACH PAGE CF BARCODE

PWIDE=8.5

TMAR IS THE HEIGHT OF THE TOP MARGIN OF THE PAGE

TMAR=1.0

ESPACE IS THE HEIGHT OF THE MARGINS REPRESENTS HOW
FAR 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

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

PERPG=IFIX((PLONG-ESPACE)/HCFACTR)

CALL PLOTS(0,0)

CALL PLOT(0,0,0,3)

ROW=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=30001(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
20 IF(MOD(IROW,PERPGE).NE.0)GOTO 30
   X=X+ESPACE
   CALL NEWPEN(2)
   XM1=X+TMAR
   XM2=X+PLONG-TMAR
   YM1=Y-SMAR
   YM2=Y+PWIDE-SMAR
   CALL PLOT(XM1,YM1,3)
   CALL PLOT(XM2,YM2,2)
   CALL PLOT(XM1,YM2,2)
   CALL PLOT(XM1,YM1,2)
   CALL PLOT(XM2,YM1,2)
   CALL PLOT(XM2,YM2,2)
   CALL SYMBOL(X,Y,TSIZE,TITLE,90.0,72)
   CALL NEWPEN(NIBS)
   X=X+TSPACE
   Y=ZERO
   CALL PLOT(X,Y,3)

LABEL THE BAR CODE ROW
30 ROWNUM=ROWNUM+1.0
   IROW=IROW+1
   CALL NEWPEN(2)
   CALL NUMBER(X,Y,SIZE,ROWNUM,90.),-11
   CALL NEWPEN(NIBS)
   X=X+CSPACE
   CALL PLOT(X,Y,3)

CONVERT THE ZERO'S AND ONE'S INTO BARS OF CORRECT WIDTH
   DO 1000 I=1,132
      CHECK FOR ZERO OR ONE BIT
         IF(IN1,EQ.ZERO)GOTO 1)
         IF(IN1,EQ.ONE)GOTO 2)
         IF(IN1,EQ.BLNK)GOTO 2000
   1000 CONTINUE
C
100
C
DRAW A ZERO BAR OF UNIT WIDTH
X=X+HEIGHT
CALL PLOT(X,Y,2)
Y=Y+DOUBLE
X=X-HEIGHT
CALL PLOT(X,Y,3)
GOTO 1000
C
C
DRAW A ONE BAR OF DOUBLE UNIT WIDTH
200
C
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
C
C
1000
CONTINUE
C
GOTO NEXT ROW
C
2000
X=X+HFACTR
Y=ZERO
CALL PLOT(X,Y,3)
GOTO 10
C
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 COMPILER.

<table>
<thead>
<tr>
<th>HP41C SOURCE CCCDE:</th>
<th>62</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>64</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>65</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>/</td>
<td>66</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X&gt;Y?</td>
<td>67</td>
<td>57</td>
<td>92</td>
<td>4</td>
</tr>
<tr>
<td>X≥Y?</td>
<td>68</td>
<td>101</td>
<td>95</td>
<td>4</td>
</tr>
<tr>
<td>X=Y?</td>
<td>69</td>
<td>54</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>6*</td>
<td>70</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6/</td>
<td>71</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>HGS+</td>
<td>72</td>
<td>50</td>
<td>43</td>
<td>4</td>
</tr>
<tr>
<td>HGS-</td>
<td>73</td>
<td>51</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>MOD</td>
<td>74</td>
<td>60</td>
<td>58</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>4+</td>
<td>76</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4-</td>
<td>77</td>
<td>63</td>
<td>61</td>
<td>3</td>
</tr>
<tr>
<td>P+</td>
<td>78</td>
<td>98</td>
<td>67</td>
<td>3</td>
</tr>
<tr>
<td>P-</td>
<td>79</td>
<td>95</td>
<td>54</td>
<td>3</td>
</tr>
<tr>
<td>LNX</td>
<td>80</td>
<td>103</td>
<td>94</td>
<td>3</td>
</tr>
<tr>
<td>SQRT</td>
<td>81</td>
<td>30</td>
<td>73</td>
<td>4</td>
</tr>
<tr>
<td>1/X</td>
<td>82</td>
<td>103</td>
<td>101</td>
<td>3</td>
</tr>
<tr>
<td>CHS</td>
<td>83</td>
<td>25</td>
<td>23</td>
<td>3</td>
</tr>
<tr>
<td>E1X</td>
<td>84</td>
<td>39</td>
<td>35</td>
<td>3</td>
</tr>
<tr>
<td>LOG</td>
<td>85</td>
<td>58</td>
<td>56</td>
<td>3</td>
</tr>
<tr>
<td>10X</td>
<td>86</td>
<td>11</td>
<td>103</td>
<td>4</td>
</tr>
<tr>
<td>10X-1</td>
<td>87</td>
<td>40</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>SINA</td>
<td>88</td>
<td>79</td>
<td>77</td>
<td>3</td>
</tr>
<tr>
<td>CCS</td>
<td>89</td>
<td>32</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>TAN</td>
<td>90</td>
<td>67</td>
<td>65</td>
<td>3</td>
</tr>
<tr>
<td>ASIN</td>
<td>91</td>
<td>15</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>ACS</td>
<td>92</td>
<td>13</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>ATAN</td>
<td>93</td>
<td>21</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>1/X</td>
<td>94</td>
<td>34</td>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>ABS</td>
<td>95</td>
<td>12</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>FXT</td>
<td>96</td>
<td>41</td>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>X#C?</td>
<td>97</td>
<td>30</td>
<td>93</td>
<td>4</td>
</tr>
<tr>
<td>XOC?</td>
<td>98</td>
<td>100</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td>LNX+X</td>
<td>99</td>
<td>57</td>
<td>55</td>
<td>5</td>
</tr>
<tr>
<td>XOC?</td>
<td>100</td>
<td>92</td>
<td>93</td>
<td>4</td>
</tr>
<tr>
<td>X#C?</td>
<td>101</td>
<td>58</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>INT</td>
<td>102</td>
<td>103</td>
<td>63</td>
<td>51</td>
</tr>
<tr>
<td>FRC</td>
<td>103</td>
<td>45</td>
<td>43</td>
<td>3</td>
</tr>
<tr>
<td>D-R</td>
<td>104</td>
<td>33</td>
<td>31</td>
<td>3</td>
</tr>
<tr>
<td>D+F</td>
<td>105</td>
<td>67</td>
<td>66</td>
<td>3</td>
</tr>
<tr>
<td>HME</td>
<td>106</td>
<td>49</td>
<td>47</td>
<td>3</td>
</tr>
<tr>
<td>HR</td>
<td>107</td>
<td>62</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>RNG</td>
<td>108</td>
<td>72</td>
<td>71</td>
<td>3</td>
</tr>
<tr>
<td>C1</td>
<td>109</td>
<td>61</td>
<td>53</td>
<td>3</td>
</tr>
<tr>
<td>C2</td>
<td>110</td>
<td>26</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>XCY</td>
<td>111</td>
<td>13</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>PI</td>
<td>112</td>
<td>36</td>
<td>92</td>
<td>2</td>
</tr>
<tr>
<td>CLIST</td>
<td>113</td>
<td>30</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>RI</td>
<td>114</td>
<td>74</td>
<td>65</td>
<td>3</td>
</tr>
<tr>
<td>RDN</td>
<td>115</td>
<td>57</td>
<td>53</td>
<td>5</td>
</tr>
<tr>
<td>LASTX</td>
<td>116</td>
<td>58</td>
<td>53</td>
<td>5</td>
</tr>
</tbody>
</table>

235
| CLX | 115 | 31 | 29 | 3 | 1 |
| AXE | 120 | 45 | 99 | 4 | 1 |
| AMA | 121 | 45 | 97 | 4 | 1 |
| XCG | 122 | 78 | 75 | 4 | 1 |
| MAE | 123 | 55 | 91 | 4 | 1 |
| SDM | 124 | 76 | 74 | 4 | 1 |
| AVN | 125 | 22 | 20 | 3 | 1 |
| CLE | 126 | 22 | 26 | 3 | 1 |
| XAD | 127 | 35 | 33 | 3 | 1 |
| GRA | 128 | 69 | 68 | 3 | 1 |
| ENTER | 129 | 36 | 37 | 3 | 1 |
| RTN | 130 | 73 | 72 | 3 | 1 |
| BEEF | 131 | 23 | 21 | 3 | 1 |
| CLA | 132 | 27 | 25 | 3 | 1 |
| ASE | 133 | 66 | 64 | 3 | 1 |
| CLR | 134 | 25 | 27 | 3 | 1 |
| AGF | 135 | 15 | 13 | 3 | 1 |
| CFF | 136 | 62 | 63 | 3 | 1 |
| FRCMP | 137 | 65 | 63 | 3 | 1 |
| ADV | 138 | 4 | 12 | 3 | 2 |
| RCL | 139 | 18 | 16 | 3 | 2 |
| ST+ | 140 | 82 | 81 | 3 | 2 |
| ST- | 141 | 83 | 82 | 3 | 2 |
| ISG | 142 | 34 | 33 | 3 | 2 |
| VME | 143 | 6 | 8 | 3 | 2 |
| CEF | 144 | 14 | 12 | 3 | 2 |
| AS | 145 | 25 | 23 | 3 | 2 |
| ARC | 146 | 36 | 34 | 3 | 2 |
| FJX | 147 | 9 | 8 | 3 | 2 |
| SCI | 148 | 44 | 43 | 3 | 2 |
| ENG | 149 | 76 | 74 | 3 | 2 |
| TONE | 150 | 33 | 32 | 3 | 2 |
| XRCM | 151 | 17 | 15 | 3 | 2 |
| XRC | 152 | 74 | 72 | 3 | 2 |
| XRC | 153 | 45 | 43 | 3 | 2 |
| XRC | 154 | 18 | 16 | 3 | 2 |
| XRC | 155 | 30 | 28 | 3 | 2 |
| XRC | 156 | 42 | 40 | 3 | 2 |
| XRC | 157 | 75 | 73 | 3 | 2 |
| SF | 158 | 77 | 75 | 3 | 2 |
| FSC | 159 | 45 | 43 | 3 | 2 |
| FC | 160 | 44 | 42 | 3 | 2 |
| FS | 161 | 60 | 58 | 3 | 2 |
| AS | 162 | 88 | 86 | 3 | 2 |
LIST OF REFERENCES


<table>
<thead>
<tr>
<th>No.</th>
<th>Distribution List</th>
<th>Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Defense Technical Information Center, Cameron Station, Alexandria, Virginia 22314</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td>Library, Code 0142, Naval Postgraduate School, Monterey, California 93940</td>
<td>2</td>
</tr>
<tr>
<td>3.</td>
<td>Department Chairman, Code 55, Naval Postgraduate School, Monterey, California 93940</td>
<td>1</td>
</tr>
<tr>
<td>4.</td>
<td>Associate Professor S. H. Parry, Code 55Py, Naval Postgraduate School, Monterey, California 93940</td>
<td>5</td>
</tr>
<tr>
<td>5.</td>
<td>Associate Professor R. H. Shuddle, Code 55Su, Naval Postgraduate School, Monterey, California 93940</td>
<td>5</td>
</tr>
<tr>
<td>6.</td>
<td>Office of the Commanding General, U.S. Readiness Command, MacDill AFB, Florida 33621</td>
<td>1</td>
</tr>
<tr>
<td>7.</td>
<td>Office of the Commanding General, U.S. Army Training and Doctrine Command, Fort Monroe, Virginia 23651</td>
<td>1</td>
</tr>
<tr>
<td>8.</td>
<td>Chief, TRADOC Research Element Monterey, Naval Postgraduate School, Monterey, California 93940</td>
<td>1</td>
</tr>
<tr>
<td>9.</td>
<td>Headquarters, U.S. Army Training and Doctrine Command, Fort Monroe, Virginia 23651</td>
<td>1</td>
</tr>
<tr>
<td>10.</td>
<td>Headquarters, U.S. Army Training and Doctrine Command, Fort Monroe, Virginia 23651</td>
<td>1</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: ATTAG-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

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: DASD-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: DRXSY-CN (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-0S (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