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.
**Title**: A Cross Compiler and Programming Support System for the HP41CV Calculator

**Author**: James Norman Richmann

**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:

Thesis Advisor:

Co-Advisor:

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
LIST OF FIGURES

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

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

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

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

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

**EDITON=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 COMPILERS AND AUTO GENERATION BAR CODE</td>
</tr>
<tr>
<td>PP21</td>
<td>PRINT</td>
<td>P</td>
<td>PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM</td>
</tr>
<tr>
<td>PP22</td>
<td>*</td>
<td>*</td>
<td>RESERVED FOR FUTURE USE BY HP41 EMULATOR</td>
</tr>
<tr>
<td>PP23</td>
<td>COMP</td>
<td>C</td>
<td>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>ERASE</td>
<td>ERASE THE SOURCE FILE, LISTING FILE AND TEXT FILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMS</td>
<td>CMS ALLOWS EXECUTION OF ANY VALID CMS COMMAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>CP ALLOWS EXECUTION OF ANY VALID CP COMMAND</td>
<td></td>
<td></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 STOP</td>
<td>S</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 HELP</td>
<td>H</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 ENTER</td>
<td>E</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 BAR</td>
<td>B</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 NEW</td>
<td>N</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 FILETYPE &quot;HP41&quot; AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.</td>
</tr>
<tr>
<td>PP18 DIREC</td>
<td>D</td>
<td>THIS COMMAND DISPLAYS THE FULL COMMAND MENU. IT HAS PRIMARY USE WHEN YOU FINISH AN OPERATION THAT PILLS 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>PP-KEY</th>
<th>CMD</th>
<th>CODE</th>
<th>ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP19</td>
<td>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</td>
<td>OCOMP</td>
<td>C 0</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</td>
<td>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</td>
<td>GO</td>
<td>G</td>
<td>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.</td>
</tr>
<tr>
<td>PP23</td>
<td>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</td>
<td>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 perform 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 economy 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
HP41CV. 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>Establishes a loop counter.</td>
</tr>
<tr>
<td>/</td>
<td>Clears x and pushes loop counter into y.</td>
</tr>
<tr>
<td>1</td>
<td>Recall the next data value.</td>
</tr>
<tr>
<td>+</td>
<td>Accumulate the sum.</td>
</tr>
<tr>
<td>0</td>
<td>Increment the loop counter.</td>
</tr>
<tr>
<td>LBL 00</td>
<td>If more data remains, branch;</td>
</tr>
<tr>
<td>RCL IND Y</td>
<td>else, quit and display sum.</td>
</tr>
<tr>
<td>GTO 00</td>
<td></td>
</tr>
<tr>
<td>RTN</td>
<td></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 alphanumeric data will result in the message "ALPHA DATA" and the program will halt.

Because there is no formal declaration of variables, the programmer writing programs for the HP41CV must use
extreme caution in managing his data set and insuring that the numbers stored and recalled by the calculator program are in fact the data elements desired. A typical example of an improper data reference occurs when a program is using indirect addressing and attempts to store or recall data from a non-existent data register. This programming error is so common that a special error message "NONEXISTENT" is provided by the calculator when this error is detected.

Indirect addressing is an important feature which gives the calculator a considerable amount of power and flexibility, but also represents an additional responsibility for the programmer to explicitly control. On the HP41CV all 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 languages. 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 differential equation combat model and the results evaluated. It accomplished the task, but slowed the execution of the program considerably (resulting in an overhead of 10.5 seconds
of extra execution time for every 100 subroutine calls) and did not significantly improve the size or legibility of the application program. Accordingly, this technique is not recommended and indirect addressing remains a task that must be treated explicitly by the application programmer.

C. ADDITIONAL CRITERIA FOR PROGRAM EVALUATION

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

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

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

1. User Friendliness

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

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

2. **Execution Speed**

The second most important criteria for a calculator
program is that it must yield results relatively quickly.
In preparing example programs for this thesis, this point
became very clear when testing two particular programs. One
program, a simultaneous differential equation combat model,
required in excess of 150 data values in order to yield
results. It should be noted that it was only with the
introduction of the HP41CV that it became feasible to con-
sider such large problems on a hand-held device. To accomo-
date the size of the model, the program was written so as to
economize on program steps at the expense of increased exe-
cution 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 33 randomly selects 10 soldiers and records the time it takes each to complete the course. The following times in minutes were recorded:

2.1 2.4 2.2 2.7 2.5
2.4 2.6 2.6 2.3 2.9

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

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

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

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

\[
\begin{align*}
\text{XEQ \, \text{ALPHA \, STAT1 \, ALPHA}} & \quad \text{(quick entry)} \\
\text{XEQ \, \text{ALPHA \, 51 \, ALPHA}} & \quad \text{(classroom use)}
\end{align*}
\]

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 SF 21), the calculator will stop after each data point is displayed, permitting you to change any value simply by entering the new value and pressing run/stop.

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

<table>
<thead>
<tr>
<th>Display</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSQD=0.521</td>
<td>Sum of Squared Deviations</td>
</tr>
<tr>
<td>SX=0.241</td>
<td>Sample Standard Deviation</td>
</tr>
<tr>
<td>SKEW=0.170</td>
<td>Skewness</td>
</tr>
<tr>
<td>KURT0=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:

\[ \text{MED} = 2.400 \text{ TO } (\text{Press R/S}) \]

\[ .. 2.500 \]

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
histogram. 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:

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

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

<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SET SIZE (nnn=16+NUMBER OF DATA POINTS)</td>
<td>XEQ &quot;SIZE 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? input R/S</td>
<td></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. R/S -or- correct value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SUMMARY STATISTICS ARE CALCULATED WHEN ALL DATA HAS BEEN ENTERED.</td>
<td>XBAR=xx R/S. mean SUMSQD=xx R/S sum of sq dev from mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STANDARD DEVIATION</td>
<td>SX=xx R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SKEWNESS</td>
<td>SKEW=xx R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KURTOSIS</td>
<td>KURT=xx R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CALCULATOR WILL AUTOMATICALLY SORT DATA POINTS. AND THEN DISPLAY:</td>
<td>PRGM STANDBY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEDIAN (note if N is even the median is not unique and an interval is displayed)</td>
<td>MED=xx R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MINIMUM</td>
<td>MIN=xx R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAXIMUM</td>
<td>MAX=xx R/S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>RNG=xx R/S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

55
**USER INSTRUCTIONS: SINGLE VARIABLE STATISTICS**

<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>USER OPTION TO ENTER NUMBER OF HISTOGRAM CELLS. NO INPUT IS REQUIRED.</td>
<td>CELL?</td>
<td>R/S or INPUT</td>
<td>N R/S</td>
</tr>
<tr>
<td>8</td>
<td>USER OPTION TO ENTER WIDTH OF HISTOGRAM CELLS. (HAS PRIORITY OVER NUMBER OF CELLS IF A WIDTH IS ENTERED.)</td>
<td>WIDTH?</td>
<td>R/S or INPUT</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CALCULATE HISTOGRAM (OUTPUT DATA ABOUT EACH CELL FROM LEFT TO RIGHT.)</td>
<td>CNT=II</td>
<td>R/S</td>
<td>CELL 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>XEQ &quot;AGAIN&quot;</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>EDIT AN INPUT VALUE AT ANY TIME PRIOR TO DATA SORT.</td>
<td>XEQ &quot;SC&quot;</td>
<td>INPUT POINT</td>
<td>WILL REMOVE POINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POINT?</td>
<td>INPUT NUMBR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X?</td>
<td>INPUT CORRECT VALUE</td>
<td></td>
</tr>
</tbody>
</table>

AFTER INPUT OF NEW VALUE CALCULATOR WILL RETURN TO DATA INPUT OR CALCULATION OF SUMMARY STATS AS APPROPRIATE.
### 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 &quot;SR&quot;</td>
<td></td>
</tr>
</tbody>
</table>
HP 41C SOURCE CODE: SINGLE VARIABLE STATISTICS

STAT1

1 LBL "STAT1" (RECOMMENDED ENTRY POINT)
2 CF 05 (SET VERIFY MODE OFF)
3 SF 26 (ENABLE AUDIO)
4 GTO "SS"

S1

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

"SS"

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

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

14 LBL 00
15 15
16 STO 04 (ESTABLISH INDIRECT ADDRESS BASE REG.)
17 STO 00 (INITIALIZE DATA ENTRY POINTER)
18 RCL 15 (NUMBER DATA POINTS (LAST PROBLEM))
19 CLE "N?" (CLEAR MEANS DATA ENTRY, NOT REVIEW)
20 FC? 06 (PROMPT)
21 1E3
22 / 1
23 +
24 STO 01 (SET UP LOOP COUNTER FOR DATA POINTS)
HP 41C SOURCE CODE: SINGLE VARIABLE STATISTICS

{( ——————————————
{ 01
——— DATA ENTRY LOOP ———
)

28 LBL 01 (INCREMENT DATA STORAGE POINTER
29 ISG 00 (INCREASE DATA STORAGE POINTER
30 LBL 02 (RECALL DATA VALUE
31 RCL IND 00 (RECALL DATA VALUE
32 "X (TEMP STORAGE FOR LABEL
33 FIX 3 (IS THIS REVIEW OF DATA PREV. ENTERED?
34 ARCL 01 (PROMPT USER FOR NEXT DATA VALUE
35 FIX 3 (STORE THE DATA VALUE
36 ASTO 03 (NO VERIFICATION OF DATA DESIRED?
37 FS? 06 (FOLLOWING IS THE VERIFICATION ROUTINE
38 GTO 03 (RECALL THE LABEL
39 "=" (RECALL THE STORED DATA
40 TONE 9 (CLEAR DATA ENTRY FLAG
41 Prompt (WILL STOP FOR DATA ENTRY IF P21 SET
42 STO IND 00 (WAS THERE NO DATA CHANGE DURING VIEW?
43 FC? 05 (IF THERE WAS A NEW VALUE, THEN RECORD
44 GTO 04 (IT AND GO BACK AND RE-VERIFY THE DATA.
45 LBL 03 (FOLLOWING IS THE STATISTICAL ACCUM.
46 CLA (STORES SIGMA X
47 ARCL 03 (STORES SIGMA X-SQUARED
48 "=" (STORES SIGMA X-CUBED
49 ARCL IND 00 (STORES SIGMA X-FOURTH-POWER
50 CF 22 (IS THIS A DATA REVIEW?
51 AVIEW (IF DATA ENTRY, INCREASE INPUT CNTR.
52 FC? 22 (AT END OF DATA ENTRY, RECALL INPUT
53 GTO 04 (COUNTER, WHICH IS A NUMBER EQUAL ONE
54 STO IND 00 (MORE THAN NUMBR POINTS
55 LBL 04
56 LBL 10
57 ST+ 10
58 X^2
59 ST+ 11
60 LASTX
61 ST+ 12
62 LASTX
63 ST+ 13
64 FS? 08
65 RTN
66 ISG 01
67 GTO 01
70 RCL 01
71 }
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
71 LBL "SM"  \{ENTRY ASSUMES X-REGISTER HAS A NUMBER
72 INT       \{1 MORE THAN NUMBER OF DATA POINTS.
73 1         \{STORES THE NUMBER OF DATA POINTS
74 -         \{CALL AN OUTPUT LABELING ROUTINE
75 STO 15    \{TEMP STORE FOR XBAR
76 MEAN      \{RECALL SIGMA X-SQUARED
77 "XBAR     \{RECALL XBAR
78 XEQ 97    \{RECALL NUMBER POINTS
79 STO 03    \{TEMP STORE FOR SUM OF SQUARED
80 RCL 11    \{DEVIATIONS ABOUT THE MEAN
81 RCL 03    \{NUMBER POINTS
82 X|2        \{CAN NOT USE SDEV FUNCTION BECAUSE OF
83 RCL 15    \{NON-STANDARD USE OF REGISTERS 12-14
84 *         \{STANDARD DEVIATION
85 -         \{SUM OF SQ DEVIATION ABOUT MEAN
86 STO 09    \{NUMBER POINTS
87 "SSQD     \{SECOND MOMENT
88 XEQ 97    \{SIGMA X-CUBED
89 RCL 15    \{SIGMA X-SQUARED
90 1         \{NUMBER POINTS
91 /         \{XBAR
92 SORT      \{NUMBER POINTS
93 "SX       \{SECOND MOMENT
94 XEQ 97    \{SIGMA X-CUBED
95 RCL 09    \{SIGMA X-SQUARED
96 RCL 15    \{XBAR
97 80         \{NUMBER POINTS
98 STO 05    \{THIRD MOMENT
99 RCL 12    \{SECOND MOMENT
100 RCL 11   \{OUTPUT THE SKEWNESS OF THE DATA
101 RCL 03   \{SIGMA X-FOURTH-POWER
102 *         \{SIGMA X-CUBED
103 3         \{XBAR
104 *         \{XBAR
105 -         \{XBAR
106 RCL 15    \{XBAR
107 /         \{XBAR
108 RCL 03    \{XBAR
109 *         \{XBAR
110 3         \{XBAR
111 y^x       \{XBAR
112 2         \{XBAR
113 *         \{XBAR
114 +         \{XBAR
115 STO 06    \{XBAR
116 RCL 05    \{XBAR
117 1.5       \{XBAR
118 y^x       \{XBAR
119 "SKEW     \{XBAR
120 XEQ 97    \{XBAR
121 RCL 13    \{XBAR
122 RCL 12    \{XBAR
123 RCL 03    \{XBAR
```

60
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

125 *
126 **
127 - RCL 03
128 X/2
129 RCL 11
130 **
131 6 **
132 +
133 RCL 15
134 /
135 RCL 03
136 4
137 Y1X
138 3
139 -
140 STO 07
141 RCL 05
142 X/2
143 /KURT=
144 XEQ 97
145 /* S? 09
146 RTN
147 /KURT=
148 XEQ 98
149 CF 00
150 RCL 15
151 2
152 /PRG
153 X=0? SP 00
154 LASTX
155 .5
156 +
157 RCL 04
158 **
159 MEE=
160 ARCL IND X
161 PC? 00
162 GTO 05
163 "-> TO
164 PROMPT
165 1
166 +
167 ARCL IND X
168 LBL 05
169 PROMPT

(XBAR
(SIGMA X-SQUARED
(NUMBER POINTS
(XBAR

(FOURTH MOMENT
(SECOND MOMENT

 OUTPUT THE KURTOSIS
 SHORT FORM WOULD NOT COMPUTE STATS
 WHICH REQUIRE SORTED DATA
 CALL A DATA SORTING ROUTINE
 INITIALIZE TEMP FLAG USED TO CHECK
 EVEN OR ODD NUMBER OF DATA POINTS

 WAS IT AN EVEN NUMBER OF POINTS?
 IF WAS EVEN NUMBER, SET FLAG.

 COMPUTING ADDRESS OF MEDIAN
 ADDRESS BASE REGISTER
 X-REG NOW HAS ADDRESS OF MEDIAN

 EVEN NUMBER POINTS IMPLIES THE MEDIAN
 NOT UNIQUE, BUT SPANS AN INTERVAL
 DISPLAY THE LEFT BOUNDARY OF MEDIAN
 X-REG POINTS TO RIGHT BOUND OF MEDIAN
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  (MIN
184 RCL 15  (TEMP STORE FOR THE RANGE
185 +  (DISPLAY THE RANGE
186 RCL IND X  (INITIALIZE TEMP FLAG TO MARK LAST BAR
187 "MAX"  (ADDRESS BASE REGISTER
188 XEQ 97  (COMPUTING INDEX LOOP COUNTER
189 RCL Z  (ADDRESS BASE REGISTER
190 -  (ADDRESS BASE REGISTER
191 STO 08  (NUMBER SET TO ADDRESS AND LOOP THRU DATA
192 "RNG"  (NUMBER POINTS
193 XEQ 97  (DEFAULT NUMBER OF BARS IS 2*LN(N)
194 CF 00  (VALUE IS ROUNDED NOT TRUNCATED
195 RCL 15  (USER HAS OPTION TO CHNG NUMBR CELLS
196 RCL 04  (RANGE
197 +  (USER HAS OPTION TO CHANGE CELL WIDTH
198 1E3  (NOW HOLDS CELL WIDTH NOT RANGE
199 /  (CELL WIDTH
200 RCL 08  (UPPER LIMIT OF CURRENT CELL COUNTED
201 +  (INITIALIZE CELL COUNTER
202 1  (NEXT DATA POINT
203 RCL 04  (CELL UPPER LIMIT
204 STO 01  (DEFAULT NUMBER OF BARS IS 2*LN(N)
205 RCL 15  (VALUE IS ROUNDED NOT TRUNCATED
206 LN  (USER HAS OPTION TO CHNG NUMBR CELLS
207 2*  (RANGE
208 STO 08  (USER HAS OPTION TO CHANGE CELL WIDTH
209 FIX 0  (NOW HOLDS CELL WIDTH NOT RANGE
210 RND  (CELL WIDTH
211 "CELL?"  (UPPER LIMIT OF CURRENT CELL COUNTED
212 PROMPT  (INITIALIZE CELL COUNTER
213 RCL 08  (NEXT DATA POINT
214 X<>Y  (CELL UPPER LIMIT
215 PROMPT  (DEFAULT NUMBER OF BARS IS 2*LN(N)
216 STO 08  (VALUE IS ROUNDED NOT TRUNCATED
217 LBL 06  (USER HAS OPTION TO CHNG NUMBR CELLS
218 RCL 08  (RANGE
219 LBL 06  (USER HAS OPTION TO CHANGE CELL WIDTH
220 RCL 08  (NOW HOLDS CELL WIDTH NOT RANGE
221 ST+ 09  (CELL WIDTH
222 0  (UPPER LIMIT OF CURRENT CELL COUNTED
223 STO 02  (INITIALIZE CELL COUNTER
224 LBL 07  (NEXT DATA POINT
225 RCL IND 01  (CELL UPPER LIMIT
226 RCL 09  (DEFAULT NUMBER OF BARS IS 2*LN(N)
227 FIX 3  (VALUE IS ROUNDED NOT TRUNCATED
228 RND
HP41C SOURCE CODE:

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

SINGLE VARIABLE STATISTICS

(DATA POINT LESS THAN UPPER LIMIT
(INCREMENT THE CELL COUNTER
(PREPARE TO LOOK AT NEXT DATA POINT
(SET FLAG FOR OUTPUT OF LAST BAR

(OUTPUT THE CELL FREQUENCY COUNT

(OUTPUT CELL BOUNDARY—LOWER LIMIT
(IS THIS THE LAST BAR ?

63
HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```plaintext
("SR") REVIEW THE DATA

250 LBL "SR"
251 SF 06
252 GTO 00

("SC") EDIT THE DATA

253 LBL "SC"
254 SF 08
255 CP 06
256 RCL 00
257 STO 05
258 RCL 01
259 STO 06
260 IF "POINT?"
261 SF PROMPT
262 STO 01
263 RCL 04
264 *+
265 STO 00
266 RCL IND 00
267 ST- 10
268 X^2
269 ST- 11
270 LASTX
271 *
272 ST- 12
273 LASTX
274 **
275 ST- 13
276 XEQ 02
277 CP 08
278 +
279 STO 00
280 RCL 06
281 STO 01
282 1
283 ISG X
284 GTO 02
285 GTO "SM"
```

...
HP41C SOURCE CODE:  SINGLE VARIABLE STATISTICS

\[
\begin{array}{c}
\{ \quad \{ 97 \} \quad \{ \quad \} \\
L B L \ 97 \\
\text{ARCL X} \\
\text{PROMPT} \\
\text{RTN}
\end{array}
\]

OUTPUT LABELING ROUTINE
SHELL SORT

A = INT (A/2)

B = 1 -- RESET LEFT SHELL BOUNDARY

STACK TABLE BEGINS:

A B
C D

INTERCHANGES FOLLOWING:

X(C) AND X(D)

GOTO 12

X<Y

GOTO 1

X<0?

GOTO 3

SECTION 1

HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS
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:

- F00 -- TEMP FLAG (USED IN EDIT AND HISTO RTNS)
- F05 -- VERIFY MODE (EVERY DATA POINT ECHOED)
- F06 -- INDICATES REVIEW OF DATA NOT QUERY MODE
- F08 -- INDICATES EDITING A DATA POINT
- F21 -- PRINTER ENABLE (STOPS CALCULATOR
  DURING A VIEW INSTRUCTION)
- F26 -- AUDIO ENABLE
SINGLE VARIABLE STATISTICS
SINGLE VARIABLE STATISTICS
APPENDIX B

LINEAR PROGRAMMING EXAMPLE

INTRODUCTION:

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

PROGRAM DESCRIPTION:

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

Subroutine "FINDQ" determines the pivot column by selecting the variable in the objective function with the most negative "price." If "FINDQ" discovers at least one
negative value in the objective function, then the tableau
column number associated with the most negative value will
be stored in register 05. Upon return from "FINDQ," the
main routine tests register 05 to see if it contains a non-
zero entry. If the entry is zero, it means that no further
pivots will improve the value of the objective function, and
the Simplex algorithm halts. If the program was in phase I
(flag 11 clear) and the value of the phase I objective func-
tion 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 "REALJB" queries the user for a column vector of pointers which indicate which variable is currently basic in each row. When
entering this vector of pointers, the user indicates artificial variables with a minus sign. Subroutine "READA" queries the user for the values in the initial Simplex tableau including the slack and surplus variables and the right hand side and objective function.

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

SAMPLE PROBLEM:

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

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

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

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

He is, however, constrained by the following factors:

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

These constraints may be expressed as:

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
</tr>
<tr>
<td>C</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>1</td>
<td>1</td>
</tr>
<tr>
<td>D</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>1</td>
<td>1</td>
</tr>
<tr>
<td>H1</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>1</td>
<td>1</td>
</tr>
<tr>
<td>H2</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>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</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>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</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>1</td>
<td>1</td>
</tr>
<tr>
<td>A1</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>1</td>
<td>1</td>
</tr>
<tr>
<td>A2</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>1</td>
<td>1</td>
</tr>
<tr>
<td>RHS</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>50</td>
<td>5</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

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:

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

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

10 \ R/S

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

7 \ R/S

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

For this example:

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

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

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

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

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

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

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

\[ \text{VALUE}=110.000 \]

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

\[ \text{XEQ ALPA ALP ALPA} \]

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>
</tbody>
</table>

etc.

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:

```
   1  2  3  4  5  6  7  8  9  10  11
A  B  C  D  H1  H2  S1  S2  A1  A2  RHS
   1  -1  -1  1  -1  1  1   1  15
   1  -1  1  1  -1 -1   1  15
   1  1  1  -1  -1   1  15
   1  -1   1  1  1   1  5
0  0  0  0  0  0  0  0  0  0  110
```

Thus, the solution may be interpreted as 15 units each for munitions A, B and C and 5 units for munition D.
<table>
<thead>
<tr>
<th>STEP</th>
<th>EXPLANATION</th>
<th>SEE</th>
<th>PRESS</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SET SIZE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(NNN=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21+M+(N+1)(M+2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHERE M=NUM ROWS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AND N=NUM COLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CALL THE PROGRAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ENTER THE NUMBER OF CONSTRAINTS</td>
<td></td>
<td>INPUT M</td>
<td>R/S</td>
</tr>
<tr>
<td></td>
<td>NUM ROW ?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ENTER THE NUMBER OF VARIABLES</td>
<td></td>
<td>INPUT N</td>
<td>R/S</td>
</tr>
<tr>
<td></td>
<td>NUM COL ?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCLUDE SLACKS, SURPLUS &amp; ARTIF.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ENTER CURRENT BASIC VARIABLE NUMBERS BY ROW</td>
<td></td>
<td>INPUT VAR #</td>
<td>R/S</td>
</tr>
<tr>
<td></td>
<td>BASIC 1 ?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ETC.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ENTER TABLEAU VALUES.</td>
<td></td>
<td>INPUT R/S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FOR MISTAKES OR TO REVIEW THE DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEE LAST STEP BELOW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TO FORCE THE CALCULATOR TO STOP AFTER EACH PIVOT.</td>
<td></td>
<td>R/S</td>
<td>SF 04 R/S</td>
</tr>
<tr>
<td>8</td>
<td>SIMPLEX COMPLETED: OPTIMAL SOLUTION FOUND.</td>
<td></td>
<td>VALUE= XX.XXX</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SIMPLEX COMPLETED: PROBLEM IS INFEASIBLE.</td>
<td></td>
<td>INFEAS</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 1</td>
<td>CLX</td>
<td>PROMPT WILL VANISH LEAVING DATA</td>
</tr>
<tr>
<td></td>
<td>WHAT ARE VALUES IN TABLEAU?</td>
<td>T1?</td>
<td>CLX</td>
<td>PROMPT WILL VANISH LEAVING DATA</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>
LINEAR PROGRAMMING

HP 41C SOURCE CODE:

```
{ "LP

1 LBL "LP
2 XEQ "INIT
3 LBL "ALP
4 XEQ "READMN
5 XEQ "READJB
6 XEQ "REALA
7 XEQ "CHECK
8 LBL 15
9 XEQ "FINDQ
10 RCL 05
11 X#0?
12 GTO 35
13 FS? 11
14 GTO 25
15 RCL 04
16 RCL 10
17 RCL 12
18 *
19 +
20 RCL 07
21 RCL IND Y
22 ABS
23 X<Y?
24 GTO 20
25 "INFEAS
26 AVIEW
27 STOP
28 LBL 20
29 SF 11
30 RCL 04
31 RCL 08
32 RCL 12
33 *
34 +
35 STO 14
36 GTO 15
37 LBL 25
38 RCL 04
39 RCL 09
40 RCL 12
41 *
42 +
43 RCL IND X
44 STO 00
45 "VALUE=
46 ARCL 00
47 AVIEW
48 STOP
49 LBL 35
50 XEQ "FINDP
51 RCL 06
52 X#0?
53 GTO 40
54 "UNBOUND
```
HP41C SOURCE CODE:  LINEAR PROGRAMMING

55 ARCL 05
56 STOP
57 LBL 40
58 XEQ "PIVOT"
59 GTO 15
HP 41C SOURCE CODE: LINEAR PROGRAMMING

FINDQ

60 LBL "FINDQ"
61 SF 01
62 0
63 STO 03
64 STO 05
65 1
66 RCL 11
67 XEQ "SETL"
68 LBL 31
69 RCL 14
70 RCL 01
71 +
72 STO 00
73 RCL IND X
74 RCL 03
75 -
76 RCL 07
77 CHS
78 X<Y?
79 GTO 38
80 FC? 11
81 GTO 37
82 RCL 15
83 RCL 01
84 +
85 RCL IND X
86 ABS
87 RCL 07
88 X<=Y?
89 GTO 38
90 LBL 37
91 RCL IND 00
92 STO 03
93 RCL 01
94 INT
95 STO 05
96 LBL 38
97 ISG 01
98 GTO 31
99 CF 01
100 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

```
101 LBL "FINDP"
102 SF 02
103 0
104 STO 06
105 RE20
106 STO 03
107 1
108 RCL 08
109 XEQ "SETL"
110 LBL 41
111 RCL 01
112 1
113 -
114 RCL 12
115 *
116 RCL 04
117 +
118 STO 00
119 RCL 05
120 +
121 RCL IND X
122 STO 02
123 RCL 07
124 X>Y?
125 GTO 48
126 RCL 00
127 RCL 12
128 +
129 RCL IND X
130 RCL 02
131 /
132 STO 00
133 RCL 03
134 -
135 RCL 07
136 CHS
137 X<Y?
138 GTO 48
139 LBL 47
140 RCL 00
141 STO 03
142 RCL 01
143 INT
144 STO 06
145 LBL 48
146 ISG 01
147 GTO 41
148 CP 02
149 RTN
```
HP41C SOURCE CODE:  LINEAR PROGRAMMING

{(-----------------------------------}
PIVOT
{-----------------------------------}

150 LBL "PIVOT"
151 SF 03
152 RCL 06
153 1
154 -
155 RCL 12
156 *
157 RCL 04
158 +
159 STO 03
160 RCL 05
161 +
162 RCL IND X
163 1/X
164 STO 00
165 1
166 RCL 12
167 XEQ "SETL"
168 LBL 51
169 RCL 03
170 RCL 01
171 +
172 RCL 00
173 ST* IND Y
174 ISG 01
175 GTO 51
176 1
177 RCL 09
178 FS? 11
179 0
180 FC? 11
181 1
182 +
183 XEQ "SETL"
184 LBL 52
185 RCL 06
186 RCL 01
187 INT
188 X=Y?
189 GTO 59
190 1
191 -
192 RCL 12
193 *
194 RCL 04
195 +
196 STO 16
197 RCL 05
198 +
199 RCL IND X
200 STO 00
201 2
202 RCL 12
203 XEQ "SETL"
HP41C SOURCE CODE

204 LBL 53
205 CP 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 ISG 02
227 GTO 53
228 LBL 59
229 ISG 01
230 GTO 52
231 RCL 13
232 RCL 06
233 *
234 RCL 05
235 STO IND Y
236 1
237 ST+ 17
238 CP 03
239 TONE 7
240 FC? 04
241 RTN
242 \"PIVOT \"
243 ARCL 17
244 PROMPT
245 RTN
246 LBL "READMN
247 7
248 STO 00
249 "NUM ROWS
250 XEQ "IN
251 XEQ "NXT
252 XEQ "NXT
253 "NUM COLS
254 XEQ "IN
255 XEQ "NXT
256 RCL 10
257 *
258 RCL 04
259 +
260 1
261 ST+ 00
262 RDN
263 STO IND 00
264 RCL 09
265 +
266 XEQ "SIZE?
267 FC?C 25
268 PROMPT
269 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

{-----
{ -----}
{ SIZE? }
{ -----}

270 LBL "SIZE?
271 "SIZE>=
272 ARCL X
273 1
274 -
275 SF 25
276 RCL IND X
277 RTN

{ -----}
{ -----}
{ "NXT " }
{ -----}

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

{ -----}
{ -----}
{ SETL }
{ -----}

284 LBL "SETL
285 1E03
286 /
287 +
288 STO IND Y
289 RTN
HP41C SOURCE CODE:  LINEAR PROGRAMMING

{--------------------------

{ READJB

}

{--------------------------

291  LBL "READJB"
292  1
293  RCL 08
294  XEQ "SETL"
295  RCL 13
296  STO 00
297  FIX 0
298  LBL 01
299  "BASIC "
300  ARCL 01
301  "IN"
302  XEQ "IN"
303  ISG 01
304  GTO 01
305  FIX 4
306  RTN

}
HP41C SOURCE CODE: LINEAR PROGRAMMING

{(-----------------------------------)
  \{ READA \}
  \{-----------------------------------\}

307 LBL "READA
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 "T
321 ARCL 02
322 FIX 4
323 LBL 13
324 RCL 01
325 1
326 -
327 RCL 12
328 *
329 RCL 02
330 +
331 RCL 04
332 +
333 1
334 -
335 STO 00
336 LBL 14
337 FC? 10
338 GTO 16
339 XEQ "IN
340 GTO 17
341 LBL 16
342 "=
343 ARCL IND 00
344 AVIEW
345 LBL 17
346 ISG 02
347 GTO 12
348 ISG 01
349 GTO 11
350 CF 10
351 0
352 RTN
HP41C SOURCE CODE: LINEAR PROGRAMMING

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

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

370 LBL "ERR1
371 0
372 LN
373 XEQ "READJB
```
HP41C SOURCE CODE:  LINEAR PROGRAMMING

{------------------------------------------}
{ CHECK }
{------------------------------------------}

374 LBL "CHECK"
375 SF 11
376 1
377 RCL 12
378 XEQ "SETL"
379 RCL 04
380 RCL 09
381 RCL 12
382 *
383 +
384 STO 14
385 STO 15
386 LBL 91
387 RCL 14
388 RCL 01
389 +
390 0
391 STO IND Y
392 ISG 01
393 GTO 91
394 1
395 RCL 08
396 XEQ "SETL"
397 LBL 92
398 CF 07
399 RCL 13
400 RCL 01
401 *
402 RCL IND X
403 X<0?
404 SF 07
405 ABS
406 STO 00
407 X=0?
408 GTO "ERR1"
409 RCL 12
410 X=Y?
411 GTO "ERR1"
412 2
413 RCL 09
414 XEQ "SETL"
415 LBL 93
416 CF 08
417 RCL 01
418 INT
419 RCL 02
420 INT
421 X=Y?
422 SF 08
423 1
424 -
425 RCL 12
426 *
427 RCL 00
HP41C SOURCE CODE: LINEAR PROGRAMMING

423 RCL 04
424 +
425 RCL 04
426 +
427 FC? 08
428 0
429 FS? 08
430 1
431 STO IND Y
432 ISG 02
433 GTO 93
434 RCL 07
435 STO IND Y
436 CF 11
437 RCL 14
438 RCL 00
439 STO IND Y
440 RCL 12
441 XEQ "SETL
442 LBL 96
443 RCL 01
444 RCL 01
445 INT
446 RCL 12
447 RCL 02
448 RCL 04
449 +
450 RCL IND X
451 CHS
452 RCL 14
453 RCL 02
454 +
455 X<>Y
456 RCL IND Y
457 +
458 STO IND Y
459 ISG 02
460 GTO 96
461 LBL 98
462 ISG 01
463 GTO 92
464 PC? 11
465 RTN
466 RCL 04
467 RCL 08
468 RCL 12
469 RCL 12
470 *
471 STO 14
472 RTN
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 = M 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 A 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.
LINEAR PROGRAMMING
"LP" LINEAR PROGRAMMING
APPENDIX C

SUBROUTINES FOR READ ONLY MEMORY

INTRODUCTION:

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

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

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

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

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

Subroutine "IN"

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

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

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

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

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

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

This subroutine is intended for use with the "ISG" loop structure which has the effect most like that of a FORTRAN "DO LOOP."

For example, to execute a loop 20 times:

```
... 20
XEQ "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"
COMMON SUBRoutines

{| 1 LBL "IN" | (INPUT MODE—DISPLAY LABEL AND ?) |
| 2 SF 10 | (SET QUERY ONCE FLAG) |
| 3 GTO 05 | |
| 4 GTO 01 | (OUTPUT MODE—DISPLAY LABEL AND DATA) |
| 5 ASTC 05 | (INSURE NO QUERY) |
| 6 RCL IND 00 | (ASSUMES ROO POINTS TO STORAGE REG) |
| 7 LBL "IX" | (DIRECT MODE—ASSUMES X REG HOLDS DATA) |
| 8 AARCL 05 | (ASSUMES LABEL SET UP BY CALLING PROG) |
| 9 LBL 01 | |
| 10 "=" | (QUERY OR DISPLAY VALUE?) |
| 11 FS? | |
| 12 FC? | 10 |
| 13 ARCL IND 00 | (DISPLAY DATA VALUE) |
| 14 CF 22 | (DIGIT ENTRY DETECTION FLAG) |
| 15 CF 22 | (DISABLE PRINTER FOR QUERY) |
| 16 (PRT: FS? 10 | |
| 17 (PRT: CF 21 | |
| 18 STC IND 00 | (STORE INPUT VALUE) |
| 19 CLA | (PREPARE ALPHA REG FOR NEXT PROMPT) |
| 20 FC? | 05 | (NOT VERIFY MODE?) |
| 21 GTC 03 | |
| 22 GTC 04 | (FOLLOWING IS ACTION TAKEN WHEN IN VERIFY MODE) |
| 23 FS? | 22 | (Any input detected?) |
| 24 FC? | 10 | (REDISPLAY VALUE IF USER INPUT) |
| 25 GTC 04 | |
| 26 GTC 04 | (DISPLAY VALUE ONCE IF NO INPUT) |
| 27 CF 10 | |
| 28 ARCL 05 | (STORE INPUT VALUE) |
| 29 CF 01 | (NOTE MUST HAVE QUERIED ONCE) |
| 30 LBL 01 | (FOLLOWING IS ACTION TAKEN FOR NON-VERIFY MODE) |
| 31 GTC 04 | (INPUT MODE?) |
| 32 GTO 04 | (IF INPUT MODE, EXIT ROUTINE) |
| 33 FS? | 22 | (WAS THERE INPUT?) |
| 34 GTC 02 | (USER CHANGED VALUE, SO VERIFY.) |
| 35 RCL 04 | (PREPARE TO EXIT) |
| 36 RCL 04 | (PRINTER ATTACHED?) |
| 37 LBL 05 | |
| 38 LBL 06 | (INCREMENT POINTER FOR NEXT IC OPERATION) |
| 39 RTN | (FINAL VALUE IS IN X REG UPON EXIT) |
| 40 RTN | (PRINT THE FINAL VALUE) |
| 41 RTN | |
| 42 RTN | |
| 43 RTN | |
| 44 RTN | |

116
HP41C SOURCE CODE: COMMON SUBROUTINES

```
35 LBL "D2" (SETUP LOOP USING REG 2 AS INDEX COUNTER
40 2
41 GTO 07
42 LBL "D1" (SETUP LOOP USING REG 1 AS INDEX COUNTER
43 1
44 LBL 07 (NUMBER LOOP ITERATIONS MUST BE IN X
45 X<>Y (PRIOR TO CALLING THIS SUBROUTINE.
46 1E3
47 /
48 1
49 +
50 STO IND Y
51 RTN
```
FP41C SOURCE CODE: COMMON SUBROUTINES

```
({-------------------})
{ }
{ VR }
{ }
{-------------------}

52 LBL "VR
53 SF C5
54 LBL "WR
55 CF 26
56 LBL "WS
57 CLRG
58 GREG 10
59 CF 29
60 IE-64
61 STC 03
62 15
63 STO 04
64 +
66 STO 00
(PRT: ADV
(PRT: SF 12
(PRT: FC? 55
(PRT: PRA
(PRT: CF 12
(PRT: ADV
67 FC? 13
68 RIN
69 X<> Z

(set verify mode on)
(initialize for classroom use -- no audio
(standard initialization routine follows
(sets display—no decimal point if integer
(effective zero level
(normal indirect address base register. (this is 1 less than the number of the
(first register where data is stored.
(double wide printing
(printer attached?
(set back to single wide print
(has initialization routine been asked to
(verify size?)
```
LBL "SZ
11 "SET SZ=
12 ARCL X
15 SF 25 (PREPARE TO IGNORE ERROR
16 RCL IND X (TEST FLAG 25 TO SEE IF SUFFICIENT
17 FC?C 25 PROMPT
18 PRCNMPT
19 RTN (SIZE EXISTS.

LBL "ER (DISPLAY "DATA ERROR" PROMPT & SOUND TONE.
21 0
22 LN (BEST WAY TO DETERMINE WHERE ERROR
23 TUNE 2 (OCCURRED IS TO HIT THE SST KEY ONCE,
24 RTN (THEN GO INTO PRGM MODE.
FP41C SOURCE CODE: COMMON SUBROUTINES

```
LBL "SCRT"  \{ RECALL NUMBER OF DATA POINTS \}
RCL 01    \{ DEFINE A = "MIDPOINT" OF NUMBER POINTS \}
LBL 00    \{ RECALL MIDPOINT \}
2
/    \{ A = INT(A/2) \}
INT 01    \{ TEST TO SEE IF LIST SORTED \}
RTN
STQ 02    \{ B = 1 -- RESET LEFT SHELL BOUNDARY \}
LBL 01    \{ STACK TABLE FOLLOWS: \}
STC 03    \{ C=B \}
STC 02    \{ C \}
RCL 01    \{ A \}
+    \{ D=C+A \}
RCL 04    \{ BASE \}
+    \{ ADDR D \}
RCL IND \{ \( X(D) \) \} X \{ D \} \{ ADDR D \} B
RCL 03    \{ X(D) \}
RCL 04    \{ C \}
RCL IND \{ X(D) \}
X>Y    \{ X(D) \}
GTO 03    \{ ADDR C \}
STC 01    \{ ADDR C \}
STC 03    \{ ADDR D \}
STC 03    \{ ADDR D \}
X>Y    \{ ADDR C \}
GTO 02    \{ ADDR C \}
STC 02    \{ ADDR D \}
RCL 01    \{ N \}
-    \{ E=N-A \}
RCL 02    \{ B \}
-    \{ B+1 \}
RCL 02    \{ E \}
+    \{ B \}
STC 02    \{ E \}
X<=Y    \{ E \}
GTO 01    \{ E \}
RTN
```

SORT USES THE FOLLOWING REGISTERS:

RO1 = A
RO2 = B
RO3 = C
RO4 = INDIRECT ADDRESS BASE
R15 = NUMBER DATA POINTS (SET BY &+)
FP41C SOURCE CODE: COMMON SUBROUTINES

{(-----------------------}
{(-----------------------}
{(-----------------------}
{(-----------------------}
SC
{(-----------------------}
{(-----------------------}

133 LBL "SC
134 RCL 00
135 1
136 +
137 STG 05
138 -1EC3
139 STG 10
140 -1
141 3
142 /
143 STG 09
FP4IC SOURCE CODE: COMMON SUBROUTINES

```c
((-------------)
  
  PUT
  
  (-------------)

144 LBL "PUT
145 "VALUE
146 11
147 STC 00
148 XEQ "TA
149 SF C2

((-------------)
  
  GET
  
  (-------------)

150 LBL "GET
151 "BIT REG
152 PKCMPT
```
CCMMON SUBROUTINES

**HP41C SOURCE CODE:**

```
COMMON SUBROUTINES

((--------------------------))
((                         ))
((                         ))
SA
((                         ))
((--------------------------))

153 LBL "SA"
154 3
155 /
156 INT
157 STC 07
158 LASTX
159 -
160 RCL 09
161 / 12
162 +
163 STO 03
164 RCL 03
165 ST+ 07
166 RCL IND 07
167 INT
168 STC 14
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?C 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 STO IND 07
200 RTN
```
HP41C SOURCE CODE: COMMON SUBROUTINES

((

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-F04 = 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 WIDE PRINTING -- LOCAL WITHIN "IO"
F13 = WHEN SET MEANS MAIN ROUTINE ASKING "VR" FOR
AUTOMATIC SIZE CHECK AFTER INITIALIZATION.
APPLICATION PROGRAM MAY USE FREELY AFTER INIT.
F14 - F20 = AVAILABLE FOR APPLICATION PROGRAM USE.

202 END

124
APPENDIX D

THE CROSS COMPILER PROGRAM AND COMMAND PROCESSOR

DESIGN METHODOLOGY:

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

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

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

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

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

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

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

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

*** 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)  ***
--- PLOT 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 SURPRESSED AFTER ONE PRINT
*** AND COMMANDS WHICH HAVE MEANING ONLY FOR VIDEO TERMINALS
*** SUCH AS FLIST, BROWSE AND XEDIT HAVE BEEN CHANGED TO THE
*** CORRESPONDING TYPEWRITER TERMINAL EQUIVALENTS SUCH AS
*** LISTFILE, TYPE AND EDIT.

***

*** FOR THE NEW USER OF THE SYSTEM, IT IS RECOMMENDED THAT
*** THIS PROGRAM BE EXECUTED SIMPLY BY TYPING THE COMMAND

HP41CV

***

*** FOR EXPERIENCED USERS, WHO HAVE NO NEED FOR THE DESCRIPTIVE
*** INSTRUCTIONS, THE FOLLOWING COMMAND IS RECOMMENDED:

HP41CV (FN) (1ST COMMAND)

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

&IF &G2 = /HP41 &GOTO -CALLER
&IF &G2 -= / &STACK &G2
&IF &G3 -= / &STACK &G3
-CALLER
&IF &G4 -= / &STACK &G4
&IF &G5 -= / &STACK &G5
CP SET PF01 IMMED PF13
CP SET PF02 IMMED PF14
CP SET PF03 IMMED PF15
CP SET PF04 IMMED PF16
CP SET PF05 IMMED PF17
CP SET PF06 IMMED PF18
CP SET PF07 IMMED PF19
CP SET PF08 IMMED PF20
CP SET PF09 IMMED PF21
CP SET PF10 IMMED PF22
CP SET PF11 IMMED PF23
CP SET PF12 IMMED PF24
CP SET PF13 IMMED PF13
CP SET PF14 IMMED PF14
CP SET PF15 IMMED PF15
CP SET PF16 IMMED PF16
CP SET PF17 IMMED PF17
CP SET PF18 IMMED PF18
CP SET PF19 IMMED PF19
CP SET PF20 IMMED PF20
CP SET PF21 IMMED PF21
CP SET PF22 IMMED PF22
CP SET PF23 IMMED PF23
CP SET PF24 IMMED PF24
CP TERMINAL LINEND OFF
CP SPool PRINTER CONT
GLOBAL TXTLIB NONIMSL CMSLIB FORTMOD2 MCD2EENH IMSLSP
STATE INSTR CODES A
&IF &RETCD1 == 0 &USERRMDE = R
&IF &INDEX GT 0 &GOTO -PROGAM
CLRSCRN
&BEAGTYPE -ENDINTRO

*** HP41C CROSS COMPILER ***

This program is used to make it easier to write, document and revise
programs for the HP41C calculator. As output, this program will
produce optical bar code for direct entry of your program into an
HP41C or HP41CV calculator.

Warning: This program is part of an ongoing research project and as
such is subject to constant revision. While there are no known
errors, the program has not been extensively tested. To insure that
any errors you detect are promptly corrected, it is important that you
submit an error report to the program proponent as soon as possible.
In order to go from a program in your head to the finished bar code, there are three main steps:

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

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 "((" 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 CONVIENT 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:
-ENDING

**********************************************************************
***
*** ESTABLISH A NEW HP41C PROGRAM SOURCE FILE. INCLUDES TITLE
*** PROMPTING.
***
**********************************************************************
-NEW
&BYTE -ENDQ
ENTER CMS FILENAME OF YOUR PROGRAM.........(PF13 OR "STOP" RETURN TO CMS)
-ENDQ
&SWITCH1 = ON
&REAL ARGS
&GOTO -CHECK
-PROGRAM
&PROGNAME = &1
&PROGTYPE = HP41
&PROGMODE = A1
&SWITCH1 = OFF
STATE &PROGNAME &PROGTYPE &PROGMODE
&IF ERETCODE = 0 &GOTO -DISPLAY
CLRSCRN
&BEGIN -ENDINTRO2
ENTER THE LABEL YOU WISH TO HAVE PRINTED AT THE TOP OF THE BAR CODE.
-ENDINTRO2
&READ ARGs
&STACK 1 &2 &3 &4 &5 &6 &7 &8 &9 &10
&STACK LBL &PROGNAME
&STACK SET TABS 1 20 25 35 45 55 65
&STACK SET TRUNC 57
&STACK
&EDIT &PROGNAME &PROGTYPE &PROGMODE
******************************************************************************
** COMMAND DISPLAY ROUTINE ******
**
******************************************************************************
-DISPLAY
&IF /&SWITCH1 = /ON &GOTO -NEW
&IF /&ASCII = /YES &GOTO -ENDDISP
CLRSCRN
&ETYPE HP41C CROSS COMPILER ........ &PROGNAME ........ EDITION=17 SEP 81
&BEGIN -ENDDISP
SELECT DESIRED COMMAND FROM THE FOLLOWING:

<table>
<thead>
<tr>
<th>PF-KEY</th>
<th>COMMAND</th>
<th>CODE</th>
<th>ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF13</td>
<td>STOP</td>
<td>S</td>
<td>GETS YOU OUT OF THE HP41C CROSS COMPILER</td>
</tr>
<tr>
<td>PF14</td>
<td>HELP</td>
<td>H</td>
<td>SHORT EXPLANATION OF HOW TO USE THE CROSS COMPILER</td>
</tr>
<tr>
<td>PF15</td>
<td>ENTER</td>
<td>E</td>
<td>INTERACTIVE PROGRAM ENTRY (NO FILE CREATED)</td>
</tr>
<tr>
<td>PF16</td>
<td>BAR</td>
<td>B</td>
<td>SUBMIT JOB FOR PHYSICAL PRODUCTION OF BAR CODE</td>
</tr>
<tr>
<td>PF17</td>
<td>NEW</td>
<td>N</td>
<td>BEGIN WORK ON A NEW PROGRAM OR NAMED SUBROUTINE</td>
</tr>
<tr>
<td>PF18</td>
<td>DIREC</td>
<td>D</td>
<td>DIRECTORY OF COMMANDS</td>
</tr>
<tr>
<td>PF19</td>
<td>LIST</td>
<td>L</td>
<td>DISPLAY NAMES OF HP41C PROGRAMS ON DISK</td>
</tr>
<tr>
<td>PF20</td>
<td>OCOMP</td>
<td>C</td>
<td>OFFLINE COMPILE AND AUTO GENERATION OF BAR CODE</td>
</tr>
<tr>
<td>PF21</td>
<td>PRINT</td>
<td>P</td>
<td>PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM</td>
</tr>
<tr>
<td>PF22</td>
<td>*</td>
<td></td>
<td>RESERVED FOR FUTURE USE BY HP41 EMULATOR</td>
</tr>
<tr>
<td>PF23</td>
<td>COMP</td>
<td>C</td>
<td>COMPILE A SOURCE LISTING ON CMS DISK</td>
</tr>
<tr>
<td>PF24</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>
</tbody>
</table>
ALLOWS EXECUTION OF ANY VALID CP COMMAND

-ENDDISP
&TYPE INPUT COMMAND:
&READ ARGS
CLRSCRN
******************************************************************************
**
** COMMAND CHECK ROUTINE
**
******************************************************************************

-CHECK
&IF /CMS = /&1 &GOTO -CMSCMD
&IF /CP = /&1 &GOTO -CPCMD
&IF /PF13 = /&1 &GOTO -EXIT
&IF /PF14 = /&1 &GOTO -HELP
&IF /PF15 = /&1 &GOTO -ENTER
&IF /PF16 = /&1 &GOTO -SUBMIT
&IF /PF17 = /&1 &GOTO -NEW
&IF /PF18 = /&1 &GOTO -DISPLAY
&IF /PF19 = /&1 &GOTO -LISTFILE
&IF /PF20 = /&1 &GOTO -OCOMP
&IF /PF21 = /&1 &GOTO -TYPE
&IF /PF22 = /&1 &GOTO -NOTYET
&IF /PF23 = /&1 &GOTO -COMPILE
&IF /PF24 = /&1 &GOTO -XEDIT
&IF /STOP = /&1 &GOTO -EXIT
&IF /HELP = /&1 &GOTO -HELP
&IF /ENTER = /&1 &GOTO -ENTER
&IF /BAR = /&1 &GOTO -SUBMIT
&IF /NEW = /&1 &GOTO -NEW
&IF /DIREC = /&1 &GOTO -DISPLAY
&IF /LIST = /&1 &GOTO -LISTFILE
&IF /OCOMP = /&1 &GOTO -OCOMP
&IF /PRINT = /&1 &GOTO -TYPE
&IF /COMPILE = /&1 &GOTO -COMPILE
&IF /XEDIT = /&1 &GOTO -XEDIT
&IF /ERASE = /&1 &GOTO -ERASE
&IF /S = /&1 &GOTO -EXIT
&IF /H = /&1 &GOTO -HELP
&IF /E = /&1 &GOTO -ENTER
&IF /B = /&1 &GOTOB -SUBMIT
&IF /N = /&1 &GOTO -NEW
&IF /D = /&1 &GOTO -DISPLAY
&IF /P = /&1 &GOTO -TYPE
&IF /C = /&1 &GOTO -OCOMP
&IF /O = /&1 &GOTO -COMPILE
&IF /X = /&1 &GOTO -XEDIT
&IF /&1 = /FILEDEF &GOTO -INNER
&IF /&1 = /DATA &GOTO -INNER
&IF /&SWITCH1 = /ON &GOTO -PROGNAM
&TYPE ?? &1 &2 &3 &4 &5 UNRECOGNIZED
-ENDCHECK &GOTO -ENDDISP

*** Process CMS or CP commands passed to this exec. **

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

*****

** List HP41C program files on CMS disk **

**

-LISTFIL
LIST * HP41 A
&GOTO -DISPLAY

**

** Compile **

**

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

**********************************************************************
** OFFLINE COMPIL E **

******************************************************************************
-OCCMP
-ERASE &PROGNAME DATA
-FILEDEF 05 DISK &PROGNAME &PROGTYPE
-FILEDEF 06 DISK &PROGNAME LISTING
-FILEDEF 04 DISK &PROGNAME DATA
-FILEDEF 02 DISK INSTR CODES &USERMODE
&type CROSS-COMPIL E BEGINS...
-HPCROSS
-PRINT &PROGNAME LISTING (UP
-STATE &PROGNAME DATA
-IF $RETCODE = 0 &GOTO -SUBMIT
-&type Compile of &PROGNAME WAS NOT SUCCESSFUL.
-&GOTO -ENDDISP
******************************************************************************
*** USING THE INTERACTIVE MODE, ENTER A NEW PROGRAM. ***
***
******************************************************************************
-ENTER
-@BEGIN TYPE -ENDDISCAUTION
CAUTION. 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-COMPI LATION WITH THE "COMP" COMMAND.

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

INPUT RESPONSE:
-ENDDISCAUTION
-@REAC ARGS
-@IF /&1 = /Y &GOTO -DISPLAY
-CLRSRN
-@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.

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.

DO YOU WISH TO ERASE THE SOURCE CODE (YES/NO)?

SUBMIT TO MVS FOR BATCH PROCESSING
&READ VARS &JN &USERID
&JN = &PIECE OF &JN 18
&USERID = &PIECE OF &USERID 14
&STACK I //&JN JOB (USERID,1011),'HP41CV BAR CODE',CLASS=A
&STACK C //1011//10111/
&STACK I // EXEC FRTXCLGP
&STACK I //FORT-SYSIN DD *
&STACK GET VERSA 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
&BEGTYPE -ENDEHELP

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" compile. The program listing is automatically printed in hard copy on the high speed printer. If the compile 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-COMPILER'S
FEEDBACK, IT IS BEST TO SIMPLY PRINT THE SOURCE CODE
CMS FILE BY IssUING THE CMS PRINT COMMAND.

PF22 GO  G  THIS COMMAND IS USED TO INVOCe 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 INVOCe 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 COMPILe.

PF24 XEDIT  X  THIS COMMAND IS USED TO INVOCe 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.

-ENDDISP
&GOTC -ENDDISP
******************************************************************************
***
**
** EXIT COMMAND PROCESSOR
******************************************************************************
-EXIT
CP SPOOL PRINTER CLOSE NOCONT
CLRSCRN
&EXIT
DONE=.FALSE.

READ(2,101) IPRIT,PAGE
READ(2,9101)(TITLE$(JJ),JJ=1,25)
9101 FORMAT(25A1)
101 FORMAT(12I5)
DO 5 I=1,NINST
  READ(2,102)(INST$(J,I),J=1,6),CCDE(I),LINST(I)
5 CONTINUE

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

LENGTH=0
DO 16 I=1,MAX
  ( I=THE INSTR NUMBER, J=CHARACTER IN INSTR)
  ATTEMPT TO READ A TEXT STRING.
  IF(IPRIT.GE.20)WRITE(6,292) UNDER
  FORMAT('I':'NEXT INSTRUCTION:','15A4,//')
  IF(IN$(T$,LT,5)) 14,12,12
16 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. ***
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(' ',14,' ',110A1)
   268 IF(IPRT.GE.10)WRITE(6,263)
      FORMAT('FOUND COMMENT CARD. NOTHING MORE DONE.')
      GOTO 16
   C
   IF NOT A COMMENT, INCREMENT THE INSTRUCTION COUNTER AND PRINT
   THE INSTRUCTION.
   13 LENGTH=LENGTH+1
   IF(MOD(LINCNT,PAGE).EQ.0)
      CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE,$)
   1 LINCNT=LINCNT+1
      WRITE(6,269)LENGTH,(T$(J),J=1,LT)$
      FORMAT(' ',14,' ',110A1)
   269 C
   C
   TRIM OFF TRAILING COMMENTS
   IF(FIND$(COMENT,1,T$,LT,LOC)) 6000,9915,9914
      LT=LOC-1
      IF(TRIM$(T$,LT)) 6000,9916,9915
      9914 CONTINUE
      GOTO 16
      9915 CONTINUE
      C
   C
   COMPIL THE TEXT INSTRUCTION.
   IF(COMPS$(T$,LT,M,M1)) 15,16,20
      ERROR=.TRUE.
      15 CONTINUE
      C
   GOTO THE FOLLOWING INSTRUCTIONS IF END OF FILE ENCOUNTERED
   WRITE(6,259)
   259 FORMAT('**************************************************************************END OF FILE**************************************************************************')
GOTO THE FOLLOWING INSTRUCTIONS IF END COMPILATION

IF(ERROR) STOP

CALL THE BAR CODE GENERATOR.

MSAVE=M1
BI=1
IBAR=BSTR$(M,M1,ITOT,TITLE$)
WRITE(6,301) ITOT

FORMAT('END HP41C CROSS COMPILE',I5,'BYTES IN TOTAL PROGRAM')

STOP

ERROR HANDLING SECTION FOLLOWS

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

WRITE(6,6003) Func$
6003 FORMAT('*** FUNCTION CALL ERROR *** ',A4)
MAIN$=0
RETURN
END
INTEGER FUNCTION AIN$(INOPER,B)

** THIS FUNCTION CONVERTS A DECIMAL NUMBER <=256 INTO A BINARY NUMBER, WITH THE VALUES OF THE BINARY DIGITS STORED IN SUCCESSIVE ELEMENTS OF AN 8 ELEMENT ARRAY OF INTEGERS. **

** THIS FUNCTION IS CALLED FROM BSTR$ IN THE HP-41CV COMPILER. **

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

0 = CONTINUE TO COMPILE

-1 = AN ERROR IN Compiling THE INSTRUCTION.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*4 FUNC$/*AIN*/
INTEGER*2 B(8)
OPERN=INOPER

CHECK FOR VALID ENTRY OPERAND
IF((OPERN.GT.255).OR.(OPERN.LT.0)) GOTO 6000

CONVERT THE FIRST BINARY DIGIT
   D1=OPERN-128
   IF(D1) 100,110,110
   B(1)=0
   GOTO 120

   B(1)=1
   OPERND=D1

CONVERT THE SECOND BINARY DIGIT
   D2=OPERN-64
   IF(D2) 200,210,210
   B(2)=0
   GOTO 230
210 \text{B(2)=1} \\
\text{OPERND=D2} \\
\text{CONVERT THE THIRD BINARY DIGIT} \\
\text{230 D3=OPERND-32} \\
\text{IF(D3) 300,310,310} \\
300 \text{B(3)=0} \\
\text{GOTO 340} \\
310 \text{B(3)=1} \\
\text{OPERND=D3} \\
\text{CONVERT THE FOURTH BINARY DIGIT} \\
\text{340 D4=OPERND-16} \\
\text{IF(D4) 400,410,410} \\
400 \text{B(4)=0} \\
\text{GOTO 450} \\
410 \text{B(4)=1} \\
\text{OPERND=D4} \\
\text{CONVERT THE FIFTH BINARY DIGIT} \\
\text{450 D5=OPERND-8} \\
\text{IF(D5) 500,510,510} \\
500 \text{B(5)=0} \\
\text{GOTO 560} \\
510 \text{B(5)=1} \\
\text{OPERND=D5} \\
\text{CONVERT THE SIXTH BINARY DIGIT} \\
\text{560 D6=OPERND-4} \\
\text{IF(D6) 600,610,610} \\
600 \text{B(6)=0} \\
\text{GOTO 670} \\
610 \text{B(6)=1} \\
\text{OPERND=D6} \\
\text{CONVERT THE SEVENTH BINARY DIGIT} \\
\text{670 D7=OPERND-2} \\
\text{IF(D7) 700,710,710}
700  B(7) = 0
    GOTO 780
710  B(7) = 1
    OPERND = D7

CONVERT THE EIGHTH BINARY DIGIT

780  DB = OPERND - 1
    IF(DB) 800, 810, 6000
800  B(8) = 0
    GOTO 1000
810  B(8) = 1

WRITE OUT CONVERSION IF NECESSARY AND RETURN

1000 IF(IPRT .GE. 20) WRITE(6, 66) INOPR, (B(I), I = 1, 8)
       FORMAT(' TRACE ', AIN$ OPERAND='I5', BINARY='I811')
       AIN$ = 0
       RETURN

ERROR HANDLING SECTION Follows

6000 WRITE(6, 6001) FUNC$
6001  FORMAT(' CONVERSION ERROR $$$ ', A4)
       WRITE(6, 6002) INOPR
       FORMAT(' ERROR IN ', AIN$ OPERAND='I5')
       AIN$ = -1
       RETURN
       END
INTEGER FUNCTION ALPHS$(A$,LA,M,M1)

C** THIS FUNCTION INTERPRETS ALPHABETIC CHARACTERS INTO HP41C KEY
C** CODES.
C** THE RETURN VALUE OF THE FUNCTION ALPHS IS SET AS FOLLOWS:
C** 0 = CONTINUE TO COMPIL
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C**
C** IMPLICIT INTEGER(A-Z)
C** COMMON/TEXT,IDIM,IPRT
C** COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C** LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C** INTEGER*2 A$(IDIM)
C** INTEGER*2 BLNK/*'QUOTE/'"/
C** INTEGER*4 FUNC$/*'ALPH$*/
C** INTEGER*4 M(1)
C** INTEGER*4 M(1)
C** INTEGER*4 M(1)
C** INTEGER*4 FUNC$/*'COMP$*/
C** INTEGER*2 C$(60)
C** INTEGER*2 C$(60)
C** INTEGER*2 C$(60)
C** INTEGER*4 C2$(60)
C** INTEGER*4 C2$(60)
C** INTEGER*4 C2$(60)
C** INTEGER*4 LC/60/
C** IF(IPRT.GE.10) WRITE(6,200)LA, (A$(I),I=1,LA)
C** FORMAT(*'TRACE',I3,*'ALPHS',I,10A1)
C** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**
C** DO 35 I=1,LA
C** IF(A$(I).NE.QUOTE) GOTO 15
C** HAVE FOUND A QUOTE MARK--DISREGARD IF 1ST INSTR,ELSE CUIT
C** IF(I.EQ.1) GOTO 35
C** GOTO 40
C** IZ=FIND$(A$(I),LA,C$,LC,LOC)
C** IF(IZ.NE.0) GOTO 20
C** WRITE(6,207)M1
C** FORMAT(*'**** INVALID CHARACTER',5X,15)
C** A$(I)=BLNK
20 IZ=1
    M(M1)=C2(IZ)
    IF(IPRT,GE,10)WRITE(6,208)M1,C$(IZ),C2(IZ),M(M1)
    FORMAT('ALPH$',I5,'ALPHA CHARACTER T75,I3)
    2 M1=M1+1
35 CONTINUE
C
40 ALPH$=0
    RETURN
C
ERROR HANDLING SECTION FollowS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT('** STRING LENGTH ERROR **',A4)
    WRITE(6,6010) LA, LB, IDIM
6010 FORMAT('LA=',',I10,' LB=',',I10,' IDIM=',',I10)
    ALPH$=-1
    RETURN
END
INTEGER FUNCTION ALP1$(A$,LA,M1)

C** STRING A$ BEGins WITH A QUOTE AND HENCE IS AN ALPHA ENTRY
C** INSTRUCTION. THIS ROUTINE COMPILES SUCH INSTRUCTIONS.
C** THE RETURN VALUE OF THE FUNCTION ALP1$ IS SET AS FOLLOWS:
C** 0 = CONTINUE TO COMPILe
C** -1 = AN ERROR IN COMPIlING THE INSTRUCTION.
C**
C** IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
COMMNY/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/
INTEGER*2 QUOTE/""/APPEND/""/
INTEGER*4 FUNC$/'ALP1$'
INTEGER*4 M(I)
IF(IPRT,GE,10) WRITE(6,200) LA,(A$(I),I=1,LA)
200 FORMAT(' TRACe 'I3,' ALP1$: 'I11D1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C C C
C
C STRIP OFF THE LEADING QUOTE
C IF(LCUT$(A$,LA,1)) 6015,6015,10
C C C
C STRIP OFF THE TRAILING QUOTE, IF ANY
C IF(A$(LA).EQ.QUOTE)LA=LA-1
C
C SET THE LENGTH OF THE INSTRUCTION
C IF(LA.LT.15)GOTO 15
WRITE(6,204)
204 FORMAT('**** ALPHA STRING TOO LONG ****')
LA=15
C
IBYTE=LA+1
M(M1)=IBYTE
IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215
FORMAT(' ALP1$,15,' LENGTH OF THIS INSTR IS',I3)
M1=M1+1

ENCOD THE TEXT LENGTH INSTRUCTION
M(M1)=240+LA
IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211
FORMAT(' ALP1$,15,' TEXT LENGTH INSTR',T75,I3)
M1=M1+1

CHECK FOR ALPHA APPEND INSTRUCTION
IF(A$(1).NE.APPEND)GOTO 50

HAVE IDENTIFIED AN ALPHA APPEND INSTRUCTION
M(M1)=127
IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
214
FORMAT(' ALP1$,15,' ALPHA APPEND CHAR',T75,I3)
M1=M1+1
IF(LCUT$(A$,LA,1)) 6015,6015,50

ENCODE TEXTUAL STRING
ALP1$=ALPH$(A$,LA,M,M1)
RETURN

ERROR HANDLING SECTION FOLLOWS
WRITE(6,6001) FUNC$
6001
FORMAT(' *** STRING LENGTH ERROR *** ',A4)
WRITE(6,6010) LA, LB, IDIM
6010
FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
ALP1$=-1
RETURN
6015
WRITE(6,6016)
6016
FORMAT(' **** INVALID OPERAND IN ALPHA ENTRY INSTR ****')
ALP1$=-1
RETURN
END
INTEGER FUNCTION ASGN$(A$,LA$,B$,LB$)

C******************************************************************************
C**
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
INTEGER*2 A$(IDIM),B$(IDIM)
INTEGER*4 FUNC$/'ASGN$/
IF(IPRT.GE.10) WRITE(6,200)LA,$(A$(I)),I=1,LA$)
200 FORMAT('< TRAC',I3,' ASGN$ : ',I10.1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
IF(LA.EQ.0) GOTO 20
10 DC 15'I'=I,LA
   B$(I)=A$(I)
15 CONTINUE
20 LB=LA
ASGN$=1
RETURN
C
C ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
6010 FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
   ASGN$=0
RETURN
END
INTEGER FUNCTION BSTR$(M,M1,TOTAL,TITLE$)

C******************************************************************************
C**  THIS FUNCTION TAKES AN ARRAY (M) OF MACHINE CODE (DECIMAL) Instructions and converts them into an array (W) of binary Instructions. It also computes the barcode checksum, and segments the array into barcode lines.
C******************************************************************************

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

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 W(133),TITLE$(133)
INTEGER*2 ALPHA(133),BLNK/'.$',/ZERO'/0',/ONE'/1'/
INTEGER*4 FUNC$/'BSTR'/
INTEGER*4 M(I),M1,W1
10 IF(IPRT.GE.10) WRITE(6,200)
200 FORMAT(' TRACE ',I3,' BSTR$ : ',)

C

INITIALIZE COUNTERS

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

C

WRITE THE TITLE TO THE BINARY CODE ARRAY

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

C

CHECK FOR END OF PROGRAM

320 IF(M(P).LE.-99) GOTO 530
EXTRACT NUMBER OF BYTES IN INSTRUCTION

IBYTE=M(P)
NBYTE=IBYTE
P=P+1

EXTRACT NEXT OPERAND OF THE INSTRUCTION

390 OPERND=M(P)
P=P+1

CONVERT OPERAND TO BINARY AND LOAD INTO ARRAY W

CHECK=CHECK+OPERND
IF(CHECK.GT.255) CHECK=CHECK-255
IF(IPRT.GE.10) WRITE(6,555)ROW,OPERND,CHECK
555 FORMAT('SEND TO AIN$ ROW: ',I3,' OPERAND: ',I6,' CHECKSUM= ',I5)
IF(AIN$(OPERND,W(WI))) 6000,420,420

IF SUCCESSFUL CONVERSION, DECREASE BYTES REMAINING
INCREMENT THE ROW COUNT, AND CHECK TO SEE IF END OF BARCODE ROW

420 IBYTE=IBYTE-1
WI=WI+8
ROW=ROW+1
IF(ROW.EQ.13) GOTO 530

CHECK TO SEE IF INSTRUCTION HAS BEEN COMPLETELY ENCODED

480 IF(IBYTE.EQ.0) GOTO 520
GOTO 390

PROCESS END OF BARCODE ROW, FIRST SAVE ENDING LOCATION IN TEMP
BARCODE ROW (THIS LOCATION WILL BE DIFFERENT DEPENDING ON
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(ITYPE NE 0) GOTO 560
TRAIL=0
GOTO 580

C CALCULATE NUMBER OF TRAILING BYTES IN BARCODE ROW

560
TRAIL=NBYTE-IMBYTE

C COMPUTE THIRD BYTE OF BARCODE ROW AND CONVERT TO BINARY


580
THIRD=(16*LEAD)+TRAIL
WI=19
CHECK=CHECK+THIRD
IF(CHECK.GT.255) CHECK=CHECK-255
IF(IPIRGE.10) WRITE(6,555) ROW,THIRD,CHECK
IF(AINS(THIRD,W(WI))) 6000,1090,1090

C 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
SECND=16+MOD(SEQNUM,16)
SEQNUM=SEQNUM+1
WI=11
CHECK=CHECK+SECND
IF(CHECK.GT.255) CHECK=CHECK-255
IF(IPIRGE.10) WRITE(6,555) ROW,SECND,CHECK
IF(AINS(SECND,W(WI))) 6000,1180,1180

C
COMPUTE FIRST BYTE OF BARCODE ROW AND CONVERT TO BINARY

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

I180 W1=3
FIRST=CHECK
IF(IPRT.GE.10) WRITE(6,555)ROW,FIRST,CHECK
IF(AIN$(FIRST,W(W1))) 6000,1220,1220

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

I220 IF(IPRT.GE.20) WRITE(6,556)
FORMAT(' END OF BARCODE ROW***********:***************')
W(1)=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. ************

************ INSERT CALL TO VERSATEC HERE. ************
CONTINUE
IF(ENDING.EQ.132) GOTO 1736
DO 1735 I=ENDBIT,132
   ALPHA(I)=BLNK
1735 CONTINUE
WRITE(4,777) (ALPHA(I),I=1,132)
777 FORMAT(66A1,/,5X,66A1)

IF REQUIRED, PRINT THE BARCODE RCW AS ZERO'S AND ONE'S ON PAPER

IF(IPRT.GE.20) WRITE(6,201)(W(I),I=1,ENDING)
201 FORMAT(' ',132I1)

SET NUMBER OF LEADING BYTES FOR NEXT ROW AND RE-INITIALIZE

LEAD=IBYTE
TOTAL=TOTAL+ROW
W1=0
IF(M(P)) 1400,480,480

SET FINAL VALUES AND RETURN

1400 BSTR$=0
   M1=1
   RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FNC$
6001 FORMAT(' '***** ERROR IN BARCODE PRODUCTION ***** ',A4)
   PM1=P-1
   WRITE(6,6101) SEQNUM,PM1,M(PM1),CHECK
6101 FORMAT(' ' ROW='',I3,' M('',I4,'') OPERAND='',I3,' CHECKSUM='',I4)
   BSTR$=-1
   RETURN
END
INTEGER FUNCTION COMP$(A$,$LA$,M,$MI$)

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
C** A TEXT STRING A$; THE ARRAY M IS THE TOTAL ARRAY OF DECIMAL
C** INTEGER KEY CODES (MACHINE INSTRUCTIONS), AND M1 IS THE
C** POSITION WHERE THE NEXT DECODED MACHINE INSTRUCTION WILL BE
C** PLACED. Thus, THE INPUT TO THIS ROUTINE IS A TEXTUAL HP41C
C** INSTRUCTION AND THE OUTPUT IS ONE OR MORE DECIMAL KEY CODES
C** PLACED APPROPRIATELY INTO ARRAY M.
C**
C** THE RETURN VALUE OF THE FUNCTION COMP$ IS SET AS FOLLOWS:
C** 1 = END STATEMENT FOUND, END COMPILATION.
C** -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
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/1/
INTEGER*2 T1S(40),T2S(40),SAV$(40),WORK$(80)
INTEGER*2 S1S$(40),S2S$(40),SS3$(40),SS4$(40),SS5$(40)
INTEGER*4 L1,L2,SAV,LWORK
INTEGER*4 LSS1,LSS2,LSS3,LSS4,LSS5
INTEGER*4 M(I),M1
INTEGER*4 FUNC$/'COMP'/
INTEGER*2 END(3),'/F','/D'/
RCL(3),'/R','/C','/L'/
QUOTE/'N'/
APPEND'/1'/
LBL(3),'/L','/B','/L'/
GTO(3),'/G','/T','/O'/
XEQ(3),'/X','/E','/Q'/
XRO(3),'/X','/R','/O'/
STO(3),'/S','/T','/O','/P','/P'/,
IND(3),'/I','/N','/D'/

COM00010
COM00090
COM00120
COM00130
COM00140
COM00150
COM00160
COM00170
COM00180
COM00190
COM00200
COM00210
COM00220
COM00230
COM00240
COM00250
COM00260
COM00270
COM00280
COM00290
COM00300
COM00310
COM00320
COM00330
COM00340
COM00350
COM00360
COM00370
COM00380
COM00390
COM00400
COM00410
COM00420
COM00430
COM00440
COM00450
COM00460
COM00470
COM00480
SET FLAGS AND INITIALIZE COUNTERS

CHECK FOR NULL STRING ENTRY INTO COMPS

CHECK FOR CATEGORY THREE SPECIAL INSTRUCTIONS

IF(EQ$(A$,LA$ RCL,3,3)) 6000,15,11
COMP$=MEM$(A$,LA,M1,MRL)
PDIGIT=.FALSE.
RETURN

IF(EQ$(A$,LA$ STO,3,3)) 6000,2,16
MAKE A QUICK CHECK FOR THE STOP INSTRUCTION
IF(A$(4),EQ$P) GOTO 65
COMP$=MEM$(A$,LA,M1,MST0)
PDIGIT=.FALSE.
RETURN

IF(EQ$(A$,LA$ LBL,3,3)) 6000,25,21
COMP$=LBL$(A$,LA,M1)
PDIGIT=.FALSE.
RETURN

IF(EQ$(A$,LA$ GTO,3,3)) 6000,30,26
COMP$=GTO$(A$,LA,M1)
PDIGIT=.FALSE.
RETURN

IF(EQ$(A$,LA$ XEQ,3,3)) 6000,35,31
31  COMP$=XEQS$(A$,LA,M,M1)
        PDIGIT=.FALSE.
        RETURN
C    
35  IF(EQS$(A$,LA,XRD,3,3)) 6000,40,36
    COMP$=XR0$(A$,LA,M,M1)
    PDIGIT=.FALSE.
    RETURN
C    
40  IF(EQS$(A$,LA,END,3,3)) 6000,45,41
    COMP$=ENDS$(A$,LA,M,M1)
    PDIGIT=.FALSE.
    RETURN
C    
CHECK FOR ALPHABETIC ENTRY INSTRUCTION.
C    
45  IF(A$(1).NE.QUOTE) GOTO 50
    COMP$=ALP$(A$,LA,M,M1)
    PDIGIT=.FALSE.
    RETURN
C    
CHECK FOR NUMERIC ENTRY INSTRUCTION.
C    
50  IF(NUMC$(A$,LA,IANSW))6000,55,51
    COMP$=DIG$(A$,LA,M,M1)
    PDIGIT=.TRUE.
    RETURN
C    
CHECK FOR CATEGORY ONE INSTRUCTION (ONE BYTE) BY LOOKING FOR BLANK
C    
55  P1=POS$(A$,LA,BLNK,1,1)
    IF(P1) 6000,65,70
C    
NC BLANK IN STRING IMPLIES HAVE FOUND ONE BYTE INSTRUCTION
C    
65  COMP$=IONE$(A$,LA,M,M1,1)
    PDIGIT=.FALSE.
    RETURN
C    

BLANK IN STRING MEANS MULTI-WORD INSTRUCTION, NOW EXTRACT PREFIX

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

CHECK FOR INDIRECT ADDRESSING

P6=POS$(A$,LA,IND,3,1)
IF(P6) 6000,80,76
INDIR=.TRUE.
IF(IPRT$GE.20)WRITE(6,235)
FORMAT(OPLENTED INDIRECT GOTO INSTRUCTION')
IF(UCUT$(A$,LA,3)) 6000,6080,80

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

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

ERROR HANDLING SECTION FOLLOWS

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

WRITE(6,6081)
6081 FORMAT( ',**** ERROR *****' )
COMP$=-1
INTEGER FUNCTION CON$(A$,LA,B$,LB,C$,LC)

STRING A$ AND STRING B$ ARE CONCATENATED AND PLACED IN C$.
IT IS FEASIBLE TO CON$(A$,LA,B$,LB$ A$,LA) OR
CON$(A$,LA$B$,LB$LB$)
IN WHICH CASE THE APPROPRIATE STRING WILL BE REPLACED.
THE NUMBER OF CHARACTERS IN THE RESULTING STRING C$ IS RETURNED
AS THE VALUE OF THE FUNCTION CON$, UNLESS THERE IS A LOSS OF
CHARACTERS IN WHICH CASE THE NUMBER OF LOST CHARACTERS IS
RETURNED AS A NEGATIVE NUMBER.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM),B$(IDIM),C$(IDIM)
IF(IPRT.GE.10) WRITE(6,200) LA,(A$(I),I=1,LA)
IF(IPRT.GE.10) WRITE(6,201) LB,(B$(I),I=1,LB)
200 FORMAT(' TRACE ',I3,' CON$: ',I10A1)
201 FORMAT(' AND ',I3,' B$: ',I10A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000

LOSS=0

DETERMINE LENGTH OF RESULT
ILC=LA+LB
IF(ILC.LE.IDIM) GOTO 20
LOSS=ILC-IDIM
ILB=LB-LOSS
ILC=IDIM
IF(IPRT.GE.05) WRITE(6,202) LOSS
202 FORMAT(' LOSS OF ',I3,' CHARACTERS DURING CONCATENATION'
GOTO 25
20 ILB=LB
25 ILA=LA
INDEX=ILC

C
MCVE B$ INTO 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)INDEX,B$(INDEX),INDEX,C$(INDEX)
207 FORMAT('MOVE B,'I3,'=',',A1,'IS NOW C,'I3,'=',',A1)
IND=INDEX-1
INDEX=INDEX-1
35 CONTINUE

MOVE A$ INTO C$

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

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

LC=ILC
65 IF(IPRT.GE.20)WRITE(6,203)LC,(C$(I),I=1,LC)
203 FORMAT('CONCAT: LC,'I3,'=','110A1')
IF(LOSS.NE.0) GOTO 70
CONS$=ILC
GOTO 75
70 CONS$=-LOSS
75 RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
6001 FORMAT('*** STRING LENGTH ERROR *** ',A4)
CCNS=-1
RETURN
END
INTEGER FUNCTION DIGT$(A$, LA, M, M1)

** THIS IS A FUNCTION THAT IS PART OF THE HP41C COMPILER. IT IS CALLED WHEN A DIGIT ENTRY INSTRUCTION IS ENCOUNTERED.

** THE RETURN VALUE OF THE FUNCTION DIGT$ 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
LOGICAL DONE, PDIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2

INTEGER*2 $A$(DIM$

INTEGER*2 PLUS/1+/1

INTEGER*2 $C$(13)/0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0'

2 INTEGER*4 LC/13/

INTEGER*4 FUNC$\cdot$DIGT$

INTEGER*4 M(1)
LOGICAL PDECIM, CHSFLG

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

200 FORMAT('TRACE ', I3, ' DIGT$ ', '110A1')

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

C

ADD A NULL INSTRUCTION BETWEEN ADJACENT DIGIT ENTRY INSTR.

C

IF(.NOT. PDIGIT) GOTO 400

C ADJACENT DIGIT ENTRY INSTRUCTION FOUND

IBYTE=1

M(M1)=IBYTE

IF(IPRT\ .GE.20) WRITE(6,212)M1,IBYTE

212 FORMAT('DIGT$', I5, ' LENGTH OF THIS INSTR IS', I3)

M1=M1+1

M(M1)=0

IF(IPRT\ .GE.20) WRITE(6,213)M1,M(M1)

213 FORMAT('DIGT$', I5, ' NULL INSTR FOR PRECEEDING DIGIT ENTRY', T75,13)

M1=M1+1
CHECK FOR DIGIT ENTRY INSTRUCTION PRECEDED BY PLUS SIGN.

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

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

CALL LCUT$(A$,LA,1)

SET THE LENGTH OF THE INSTRUCTION

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

DO 35 I=1,LA
15 IZ=FULD$(A$(I),LA,C$,LC,LOC)
IF(I$.$NE.'0')GOTO 20
WRITE(6,207)M1
207 FORMAT('________INVALID CHARACTER________',5X,I5)
DIGITS=-1
RETURN
20 M(M1)=IZ+15
IF(IPRT.GE.10)WRITE(6,208)M1,C$(IZ),M(M1)
208 FORMAT('DIGIT$',I5,'DIGIT ENTRY INSTR',3X,A1,3X,
'T75',I3)
M1=M1+1
35 CONTINUE

DIGITS=0
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001)FUNCS
6001 FORMAT('*** STRING LENGTH ERROR ***',A4)
6010 WRITE(6,6010) LA,LB,IDIM
   DIGT$=-1
   RETURN
6999 WRITE(6,602) M1
602 FORMAT(' ',5X,'**DIGIT ENTRY INSTR ERROR**',5X,15)
   DIGT$=-1
   RETURN
   END
INTEGER FUNCTION END$(A$,LA,M,M1)
C**********************************************************************
C** STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN END INSTRUCTION.  
C** THE RETURN VALUE OF THE FUNCTION END$ IS SET AS FOLLOWS:  
C** 0 = CONTINUE TO COMPILE  
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION.  
C**********************************************************************
C 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
C C C COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C THIS SUBROUTINE PASSES THE NUMBER OF ELEMENTS IN M VIA COMMON /CNT
C
C LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C INTEGER*2 A$(IDIM)
C INTEGER*2 BLNK/*
C INTEGER*4 FUNC/* END*/
C INTEGER*4 M(I)
C IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C 200 FORMAT(' TRACE',I13,' END$:',110A1)
C IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C END INSTRUCTION IS THREE BYTES LONG.  INDICATE LENGTH OF INSTR.
C
C M(M1)=3
C IF(IPRT.GE.20)WRITE(6,201)M1,IBYTE
C 201 FORMAT(' END$',I5,' LENGTH OF NEXT INSTR IS',I3)
M1=M1+1
C
C LOAD PREFIX CODE FOR END INSTRUCTION.  193 IS USED INSTEAD OF 192.  
C WHILE 192 IS THE HP STANDARD OP-CODE FOR "END", THE 193 IS USED TO
C ENABLE 192 TO STAND FOR AN ALPHA LABEL INSTRUCTION.  THIS USAGE
C IS STANDARD AMONG THE HP USERS GROUP PRACTICING SYNTHETIC PROGRAM-END00480
C    MING.
C    M(M1)=193
      IF(IPRT.GE.10)WRITE(6,210)M1,M(M1)
210    FORMAT(' END$',15,' END INSTR',T75,I3)
      M1=M1+1
C    PROVIDE TWO NULL INSTRUCTIONS TO RESERVE SPACE FOR THE LINK
C    POINTERS. ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
C    POINTERS WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN. THIS
C    CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
C    BOUNDARIES WITHIN THE HP41CV MEMORY. THE CHAIN OF LABELS IS
C    RECOMPILED BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE
C    CHAIN ARE SET TO ZERO BY THIS COMPILER.
      M(M1)=0
      IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211    FORMAT(' END$',15,' TRAILING NULL INSTR',T75,I3)
      M1=M1+1
      M(M1)=0
      IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212    FORMAT(' END$',15,' POINTER WILL BE RECOMPILED',T75,I3)
C    NOTE NUMBER OF ELEMENTS IN THE MACHINE CODE ARRAY AND SET END FLAG
      S2=M1
      M1=M1+1
C    DONE=.TRUE.
      WRITE(6,202)S2
202    FORMAT('1COMPILATION ENDED$',15,' MACHINE CODES GENERATED')
      END$=1
      RETURN
C    ERROR HANDLING SECTION follows
C    6000 WRITE(6,6001) FNC$
6001    FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010    FORMAT(' LA=',I10,', LB=',I10,' IDIM=',I10)
      END$=-1
      RETURN
      END
INTEGER FUNCTION EQ$(A$,LA,B$,LB,NUM)

**-----------------------------------------------------------------------**
** EQ$00010**
C**
C**
C** THIS FUNCTION TESTS FOR STRING EQUALITY. FOR THIS FUNCTION ** EQ$00020**
C** THE RETURN VALUE IS CRUCIAL AS IT CONTAINS THE RESULTS OF THE ** EQ$00030**
C** TEST FOR EQUALITY.
C**
C** NUM DEFINES THE NUMBER OF CHARACTERS TO BE EXAMINED FOR ** EQ$00040**
C** EQUALITY, STARTING FROM THE LEFT MOST POSITION OF BOTH ** EQ$00050**
C** STRINGS. THEREFORE, THE STRINGS:
C**
C** A$='ABCDEF' AND B$='ABC'
C**
C** WILL BE "EQUAL" IF TESTED WITH EQ$(A$,LA,B$,LB,LB)
C** BUT "UNEQUAL" IF TESTED WITH EQ$(A$,LA,B$,LB,LA)
C**
C** TO TEST FOR ABSOLUTE EQUALITY, JUST ASSIGN NUM TO BE SOME ** EQ$00070**
C** ARBITRARILY LARGE INTEGER, SAY 100. THE COMPARISON WILL ** EQ$00080**
C** TERMINATE APPROPRIATELY AT THE END OF THE SHORTEST STRING.
C**
C** IE. EQ$(A$,LA,B$,LB,RE) WILL TEST ABSOLUTE EQUALITY
C**
C** IT IS SUGGESTED THAT THIS ROUTINE BE USED IN AN ARITHMETIC ** EQ$00090**
C** IF STATEMENT OF THE FORM:
C**
C** IF(EQ$(A$,LA,B$,LB,LA)) 6002, 10, 20
C**
C** WHERE: 6002 IS AN ERROR HANDLING ROUTINE
C** 10 IS THE ROUTINE WHEN STRINGS ARE NOT EQUAL
C** 20 IS THE ROUTINE WHEN STRINGS ARE EQUAL
C**
C**-----------------------------------------------------------------------**
C**
C** COMMON/TEXT,IDIM,IPRT
C** INTEGER*2 A$(IDIM),B$(IDIM)
C** INTEGER*4 FUNC$/'EQ$/
C** INTEGER*4 LA, LB, NUM
C**
C** IF(IPRT.GE.10) WRITE(6,230)LA,NUM,A$(I),I=1,LA
C** FORMAT(*,230)*,I3, 'EQ$(',I3, ',',I3,'=*110AI')
C** IF(IPRT.GE.10) WRITE(6,199)LB, (B$(I),I=1,LB)
C** FORMAT(*,199)*,I3, 'LB',B$(I),'=110AI')
C** IF(LA.GT.IDIM) WRITE(6,600)GOTO 6000
C** IF(LB.GT.IDIM) WRITE(6,600)GOTO 6000
C**
C** LENGTH=NUM
C**
C**-----------------------------------------------------------------------**

**-----------------------------------------------------------------------**
** EQ$00340**
C**
C**
C** COMMON/TEXT,IDIM,IPRT
C** INTEGER*2 A$(IDIM),B$(IDIM)
C** INTEGER*4 FUNC$/'EQ$/
C** INTEGER*4 LA, LB, NUM
C**
C** IF(IPRT.GE.10) WRITE(6,230)LA,NUM,A$(I),I=1,LA
C** FORMAT(*,230)*,I3, 'EQ$(',I3, ',',I3,'=*110AI')
C** IF(IPRT.GE.10) WRITE(6,199)LB, (B$(I),I=1,LB)
C** FORMAT(*,199)*,I3, 'LB',B$(I),'=110AI')
C** IF(LA.GT.IDIM) WRITE(6,600)GOTO 6000
C** IF(LB.GT.IDIM) WRITE(6,600)GOTO 6000
C**
C** LENGTH=NUM
C**
C**-----------------------------------------------------------------------**
IF((LENGTH.LE.LA), AND,(LENGTH.LE.LB)) GOTO 10
    IF(LA.NE.LB) GOTO 5
        LENGTH=LA
    GOTO 10

        STRINGS 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(6,202)
        RETURN

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

10    DO 15 I=1,LENGTH
        IF(IPRT.GE.30) WRITE(6,201) I, A$(I), I, B$(I)
        IF(A$(I).EQ.B$(I)) GOTO 15
        EQ$=0
        IF(IPRT.GE.20) WRITE(6,202) I
        FORMAT(' STRINGS FOUND UNEQUAL','I3', 'POSITION')
        RETURN
15    CONTINUE

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

        EQ$=1
        IF(IPRT.GE.20) WRITE(6,203)
        FORMAT(' STRINGS FOUND EQUAL')
        RETURN

        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)
        EQ$=-9999
        RETURN
END
INTEGER FUNCTION FIND$(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 FIND$ IS SET TO

LOC (LOCATION OF FIRST MATCH IN B$) IF MATCH FOUND

-1 IF AN ERROR IS ENCOUNTERED.

**IMPLICIT INTEGER(A-Z)**

COMMON/TEXT/IDIM, IPRT

INTEGER*2 A$(1), B$(IDIM)

INTEGER*2 OBJECT

INTEGER*4 FUNC$/'FIND$

INTEGER*4 LOC, FIND$

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

200 FORMAT(' ', I3, ' FIND$: ', I110A1)

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

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

C

OBJEES=A$(1)

INDEX=1

DO 25 I=1, LB

25 CONTINUE

NO MATCH FOUND

LOC=0

FIND$=0

IF(IPRT.GE.20) WRITE(6,201) LOC
201 FORMAT( 'NO SINGLE CHARACTER MATCH FOUND',I2) RETURN
C
C HAVE FOUND A MATCH
30 LOCC=INDEX FINDS=INDEX IF(IPRT.GE.30)WRITE(6,202)LOC
202 FORMAT( 'HAVE FOUND SINGLE CHARACTER MATCH AT',I3) RETURN
C
C ERROR HANDLING SECTION FOLLOWS
C 6000 WRITE(6,6001) FUNC$
6001 FORMAT( '*** STRING LENGTH ERROR *** ',A4) WRITE(6,6010) LA, LB, IDIM
6010 FORMAT( 'LA=',I10,' LB=',I10,' IDIM=',I10) FINDS=-1 RETURN END
INTEGER FUNCTION GTO$(A$,LA, M, M1)

**STRING A$ HAS BEEN IDENTIFIED TO CONTAIN A GTO INSTRUCTION.**

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

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

**IMPCLICT INTEGER(A-Z)**

COMMON/TEXT/IDIM, IPRI
COMMON/FLAGS/DONE, PDIGIT, PALPHA, DIGIT, ALPH, INDIR, FLAG1, FLAG2
COMMON/CNTR/P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2
LOGICAL DONE, PDIGIT, PALPHA, DIGIT, ALPH, INDIR, FLAG1, FLAG2
INTEGER*4 P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/
INTEGER*2 QUOTE/"/
INTEGER*2 IND(3) /1/, "N", "D" /
INTEGER*4 FUNC$"GTO"/
INTEGER*4 M(1)
IF(IPRT.GE.10) WRITE( 6, 200) LA, (A$(I), I=1, LA)
200 FORMAT( 12, TRACE$"I3", GTO$: "I10A1")
IF(LA.GT.IDIM.OR.LA.LE.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**
IF(A$(1)=QUOTE) GOTO 80

PROCESS LOCAL LABELS, FIRST CHECK FOR INDIRECT GTO INSTR
P1=POS$(A$,LA,BLNK,1,1)
IF(P1) 6015,20,15

PROCESS GTO INDIRECT INSTRUCTION.
IF(EQ$(A$,LA,IND,3,3)) 6015,6020,16
CALL LCUT$(A$,LA,P1)
IF(IPRT.GE.20)WRITE(6,235)
235 FORMAT(' DETECTED INDIRECT GOTO INSTRUCTION'
INDIR=.TRUE.
BYTE=2
PREFIX=174

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

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
CONTINUE

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

HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-
AND. THEN GOTO PROCESS A THREE BYTE INSTRUCTION.
INDEX=INDEX+101
GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

IF (IVAL$(A$, LA, INDEX)) 

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

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

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

IBYTE=2
M(M1) = IBYTE
IF (IPRT.GE.20) WRITE(6,210) M1, IBYTE
M1 = M1 + 1

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

M(M1) = 177 + INDEX
IF (IPRT.GE.10) WRITE(6,213) M1, M(M1)
FORMAT( ' GTO$$',I5,' TWO BYTE GTO INSTR',I75,I3)
M1 = M1 + 1

LOAD NULL INSTR FOR TWO BYTE GTO INSTR AND RETURN

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

PROCESS THE GOTO INSTRUCTION (OPERAND>14)
75  
M(M1)=IBYTE
IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
   FORMAT('GTO$','I5',' LENGTH OF NEXT INSTR IS','I3')
   M1=M1+1

LOAD THE GTO INSTR PREFIX

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

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

C
C
C
IF(INDIR) GOTO 95
   M(M1)=0
IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)
   FORMAT('GTO$','I5',' NULL FOR GTO INSTR','T75','I3')
   M1=M1+1

LOAD THE 2D OPERAND OF THE GTO INSTR

C
C
C
IF(INDIR) INDEX=INDEX+128
   NOTE THAT FOR GTO IND THE HIGH ORDER BIT IS NOT SET
   M(M1)=INDEX
IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
   FORMAT('GTO$','I5',' GTO 2D OPERAND ','T75','I3')
   M1=M1+1
   GTO$=0
   RETURN

C
C
C
******************************************************************************
******************************************************************************

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE

C
C
C
K=0
P2=PO$$('A$','LA','QUOTE','1','2)
   IF(P2) 6015,120,85
CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS

LEFT=LA-P2
IF(LEFT) 6015,100,6015

DELETE THE ENDING QUOTE BY TRUNCATING THE STRING ONE CHAR

LA=LA-1

SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
(LINE 120 ACCOUNTS FOR BEGINNING QUOTE STILL ON STRING)

LENGTH=LA-1 FOR LBL H=4 FOR XEQ H=2
H=2
IBYTE=H+LENGTH
M(M1)=IBYTE
IF(IPRT.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(IPRT.GE.10)WRITE(6,214)M1,M(M1)
214 FORMAT(" GTO$",I5," ALPH GTO INSTR",T75,I3)
M1=M1+1

SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL

U=240
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(" GTO$",I5," LENGTH CODE ALPH GTO",T75,I3)
M1=M1+1
ADD ALPHABETIC CHARACTERS AND RETURN

140   LA=LENGTH+1
      GT0$=ALPH$(A$,LA,M,M1)
      RETURN

ERROR HANDLING SECTION FOLLOWS

6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA, LB, IDIM
5010 FORMAT(' LA='',I10,' LB='',I10,' IDIM='',I10)
      GT0$=-1
      RETURN

6015 WRITE(6,6016)
6016 FORMAT(' **** INVALID SECOND OPERAND IN GTO INSTR ****')
5016 RETURN

6020 WRITE(6,6021)
6021 FORMAT(' **** FOUND THREE OPERANDS, EXPECTING IND ****')
5021 GT0$=-1
      RETURN
      END
INTEGER FUNCTION IN$(A$,LA,IN$)

C***********************************************************************
C** STRING A$ IS READ FROM UNIT IN.
C** THE LENGTH OF A$ IS AUTOMATICALLY COMPUTED, NOT COUNTING ANY
C** LEADING OR TRAILING BLANKS, WHICH ARE TRIMMED AWAY.
C** THE INPUT READER ASSUMES AN FIXED LENGTH INPUT RECORD OF 80
C** CHAR.
C***********************************************************************
IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK(/)*
INTEGER*2 CARD(80)
INTEGER*4 FUNC$/"IN"/
LOGICAL EOFILE(IO)/10*.FALSE./
FORMAT(80A1)

100 CHECK FOR END OF FILE
   IF(.NOT.Eofile(IN$)) GOTO 5
      IN$=-1
      LA=0
      IF(IPRT.GE.20) WRITE(6,201) IN$,,
      FORMAT(' ',10*ATTEMPT TO READ AFTER END OF FILE ON UNIT ',I2)
      RETURN

201 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)
   FORMAT(' ',78A1)

58 CHECK CARD FOR TRAILING BLANKS
   IM=0
   DO 60 I=1,80
      INDEX=81-I
      IF(CARD(INDEX).NE.BLNK) GOTO 65
   60 CONTINUE
IM=IM+1
60 CONTINUE
65 IF(IM.EQ.0) GOTO 70
    IF(IPRT.GE.20) WRITE(6,207) IM
    FORMAT(' FOUND',I3,' TRAILING BLANKS IN INPUT STRING')
    IF(IM.NE.80) GOTO 70
    LA=0
70 IN$=0
    IF(IPRT.GE.20) WRITE(6,208)
    FORMAT(' FOUND INPUT STRING IS ALL BLANKS')
    IF(IPRT.GE.20) WRITE(6,200)LA,(A$ (I),I=1,LA)
    RETURN

IEND=80-IM
CHECK CARD FOR LEADING BLANKS
IM=0
10 DO 15 I=1,IEND
    IF(CARD(I).NE.BLNK) GOTO 15
    IM=IM+1
15 CONTINUE
10 IF(IM.EQ.0) GOTO 25
    IF(IPRT.GE.20) WRITE(6,211) IM
    FORMAT(' FOUND',I3,' LEADING BLANKS IN INPUT STRING')
    IBEG=I+IM

Determine length of input string
LA=IEND-IBEG+1
IF(LA.LE.IDIM) GOTO 30
    LOSS=LA-IDIM
    IF(IPRT.GE.10) WRITE(6,216) LOSS
    FORMAT(' STRING TOO LONG FOR MAX STRING LENGTH. LOST',I3)
    LA=IDIM
    IEND=IEND-LOSS

TRANSFER THE CARD CHARACTERS TO THE INPUT STRING.
INDEX=1
30 DC 85 I=IBEG,IEND
    A$ (INDEX)=CARD(I)
IF(IPRT.GE.30) WRITE(6,209),I,CARD(I),INEX$=INDEX$  
209   FORMAT(' MOVE CARD(',I3,')=",A1," IS NOW C(',I3,')="A1")  
INDEX=INDEX+1  
85 CONTINUE  
C  
C CHECK FOR STRING ERROR AND RETURN  
C  
IF(IPRT.GE.20) WRITE(6,200)LA,(A$(I),I=1,LA)  
200   FORMAT(' TRACE ',I3,IN$=I10A1)  
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000  
IN$=LA  
RETURN  
C  
C HANDLE END OF FILE CONDITION  
C  
999 ECFILE(IN)=.TRUE.  
IN$=-1  
LA=0  
IF(IPRT.GE.20) WRITE(6,215)  
215   FORMAT(' END OF FILE ENCONERED')  
RETURN  
C  
C ERROR HANDLING SECTION FOLLOWS  
C  
6000 WRITE(6,6001) FUNC$  
6001 FORMAT(' *** STRING LENGTH ERROR *** ",A4)  
IN$=-1  
RETURN  
END
**INTEGER FUNCTION IONE$(A$,LA,M1,IBYTE)**

***THIS FUNCTION IS THE TABLE DRIVEN INSTRUCTION LOOKUP. IT IS***

***USED TO TRANSLATE THE ONE BYTE INSTRUCTIONS IN THE HP41C***

***COMPILED.***

***IMPLICIT INTEGER(A-Z)***

COMMON/TEXT/IDIM,IPRT

COMMON/FLAGS/DONE,PDIGIT,ALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2

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

COMMON/TABLE/INST$,LINST,CODE,NINST

INTEGER*2 INST$(6,111)

INTEGER*4 LINST(111),CODE(111),NINST

INTEGER*2 A$(IDIM)

INTEGER*4 FUNC$$/IONE$

INTEGER*4 M1)

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

200 FORMAT(' TR: ',I3, ' IONE$: ',110AI)

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

DO 50 I=1,NINST

LENGTH=LINST(I)

IF(LA.NE.LENGTH) GOTO 50

DO 30 J=1,LENGTH

IF(A$(J).NE.INST$(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(I).LE.143) GOTO 35

IF(IBYTE.EQ.2.AND.CODE(I).GT.143) GOTO 35

GOTO 6020

LOAD CORRECTLY MATCHING VALUES TO MACHINE CODE ARRAY

35 M(M1)=IBYTE

IF(IPRT.GE.20)WRITE(6,212)M1,IBYTE
212 FORMAT('IONES$','I5','LENGTH OF NEXT INSTR IS','I3')
    M1=M1+1
    M(M1)=CODE(I)
    IF(IPRT.GE.10)WRITE(6,210)M1,(INST$(JJ,I),JJ=1,6),M(M1)
210 FORMAT('IONES$','I5','I6A1','INSTR','I75','I13')
    M1=M1+1
    C IF(LA.LT.LENGTH) GOTO 35
       LA=0
       IONES$=0
       RETURN
    C
    C FOLLOWING COMMENT LINES HAVE BEEN RETAINED TO FACILITATE USE OF
    C PROGRAM UNDER RULE THAT THE LENGTH OF A$ MAY BE MORE THAN THE
    C LENGTH OF THE MATCH STRING. CODE HAS BEEN TESTED AND PROVEN TO
    C SELECT FIRST SUBSTRING MATCH IN TABLE.
    C
    C35
    LEFT=LA-LENGTH
    LSTART=LENGTH+1
    IF(SEG$(A$,LA,A$,LA,LSTART,LEFT).GT.0) GOTO 40
       IONES$=0
       RETURN
    C
    C40
    IONES$=LA
    RETURN
    C
    C INSTRUCTION DOES NOT MATCH SO CHECK NEXT INSTRUCTION
    C
    50 CONTINUE
    C
    C NO MATCH FOUND
    C
    60 WRITE(6,215)
215 FORMAT('**** UNRECOGNIZABLE INSTRUCTION ****')
    M(M1)=IBYTE
    IF(IPRT.GE.20)WRITE(6,212)M1,IBYTE
    M1=M1+1
    M(M1)=0
    IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211 FORMAT('IONES$','I5','NULL INSTR','I75','I13')
    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)
       WRITE(6,6010) LA, LB, IDIM
6010   FORMAT('** LA=',I10,**, 'LB=',I10,**, 'IDIM=',I10)
       ICNES=-1
       RETURN
6020   WRITE(6,6021)
6021   FORMAT('*** LENGTH OF INSTRUCTION DOES NOT MATCH NUMBER OPERNDS')
       ICNES=-1
       RETURN
       END
INTEGER FUNCTION IRDR$(IDEV, VAL)

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

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 A$(80)
INTEGER*2 BLNK, 'ZERO', '0', 'MINUS', '-'/
DATA LL/256/
INTEGER*4 RVAL, SIGN, IFN, FRAC
INTEGER*4 IRDR$, VAL, IDEV
IF(IPRT.GE.20)WRITE(6,200)IDEV

200 FORMAT(* TRACE IRDR$ *,15)

C
C
C
IF(IN$(A$,LA,IDEV))6000,12,12

C
C
12 SIGN=1
IFN=0
15 DO 1 II=1,LA
IF(IPRT.GE.30)WRITE(6,213)A$(II)
213 FORMAT(* EXAMINING *,A1,* FOR INTEGER VALUE*)
IF(A$(II).EQ.MINUS)GO TO 10
IF(A$(II).EQ.BLNK)GO TO 1
20 ITEMP=A$(II)-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*)
217 CONTINUE
217 FORMAT(* SUBROUTINE IRDR$ RETURNING VALUE ',',I20)
IF(IPRT.GE.30)WRITE(6,2225)IFN,SIGN
2225 FORMAT(* SET FINAL SIGN OF ',',I20, ' WITH', ',',I20)

660 IRDR$=SIGN*IFN
VAL=IRDR$
IF(IPRT.GE.20) WRITE(6,217) VAL
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001)
FORMAT('*** END OF FILE DETECTED *** ','A4)
IRDR$=-1
STOP
WRITE(6,6008)(A$(1),I=1,LA)
FORMAT('*** ATTEMPT TO FIND INTG VALUE OF ALPHABETIC STRING: ','110A1)
STOP
END
INTEGER FUNCTION ITWO$(A$,LA,M1,INDIR)
C*************************************************************************
C** STRING A$ IS A POSTFIX FOR A MULTI-WORD INSTRUCTION.
C** THIS ROUTINE WILL EXAMINE THE POSTFIX AND RETURN A DECIMAL
C** VALUE INTERPRETATION OF THE POSTFIX.
C** INDIRECT INSTRUCTIONS WILL HAVE THE POSTFIX APPROPRIATELY
C** SET WITH THE HIGH ORDER BIT ON, AS REQUIRED BY THE INDIR FLAG
C** THE RETURN VALUE OF THE FUNCTION ITWO$ IS SET AS FOLLOWS:
C** 0 = CONTINUE TO COMPIL
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C*************************************************************************
IMPLICIT INTEGER(A-Z)
COMMON/TEXT,IDIM/IPRT
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/
INTEGER*2 LABEL(26) /'A$','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$','/'
2
3
INTEGER*4 FUNCS'/ITWO$'/
INTEGER*4 M(1)
LOGICAL INDIR
IF(IPRT.GE.10) WRITE(6,200)LA, {A$(I),I=1,LA)
F FORMAT('TRACE: ',13,'ITWO$: ',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
CHECK FOR BLANK INDICATING A BOGUS THIRD OPERAND
IF(TRIM$(A$,LA)) 6015,6030,10
IF(POS$(A$,LA,BLNK,1,1)) 6015,20,6020
C
CHECK FOR NUMERIC OPERAND
IF(NUMC$(A$,LA,IANSW)) 6015,25,50
C OPERAND MUST BE REGISTER X,Y,Z,T OR L OR A LOCAL ALPHA LABEL

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

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

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

35 INDEX=INDEX+101
   GOTO 75

C OPERAND MUST BE A NUMERIC LOCAL LABEL

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

C HAVE FOUND VALID POSTFIX, CHECK FOR INDIRECT INSTRUCTION

75 IF(INDIR) INDEX=INDEX+128

C LOAD THE SECOND OPERAND IN THE MACHINE CODE ARRAY

95 M(M1)=INDEX
   IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
   FORMAT('ITWO$',15,'2D OPERAND ',T75,I3)
   M1=M1+1

C CLEAN-UP AND RETURN

C INDIR=.FALSE.
   ITWO$=0
RETURN

C
C
C ERROR HANDLING SECTION FOLLOWS
C
C 6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)$
WRITE(6,6010) LA, LB, IDIM
6010 FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
ITWOS=-1
RETURN
6015 WRITE(6,6016)
6016 FORMAT(' ***** INVALID SECOND OPERAND IN INSTR *****')
ITWOS=-1
RETURN
6020 WRITE(6,6021)
6021 FORMAT(' ***** FOUND THREE OPERANDS, EXPECTING IND *****')
ITWOS=-1
RETURN
6030 WRITE(6,6031)
6031 FORMAT(' ***** FOUND SECOND OPERAND BLANK ******')
ITWOS=-1
RETURN
END
INTEGER FUNCTION IVAL$(A$, LA, VAL)

CONVERTS A NUMERIC TEXT STRING TO INTEGER NUMERIC VALUE.

IMPLICIT INTEGER(A-Z)
COMON/TEX/IDIM, IPRT
INTEGER*2 A$(IDIM)
INTEGER*2 BLNK/*",ZERO/'0",MINUS/'-
INTEGER*4 FUNC$*/'VAL/"
DATA LL/256/
INTEGER*4 RVAL, SIGN, IFN, FRAC
INTEGER*4 IVAL$, VAL
IIF(IPRT GE 10) WRITE(6, 200) A$(I), I=1, LA)
FORMAT(* TRACI*, I3, * IVAL$, I110A1)
IF(LA*, GT, IDIM OR, LA*, LT, 0) GOTO 6000

SIGN=1
IFN=0
15 DC 1 II=1, LA
IIF(IPRT GE 30) WRITE(6, 213) A$(I)
FORMAT(* EXAMIMING 'A', FOR INTEGER VALUE*)
IF(A$(I), EQ, MINUS) GOTO 10
IF(A$(I), EQ, BLNK) GOTO 1
ITEMP=A$(II)-ZERO
ITEMP=ITEMP/LL
IF((ITEMP, GT, 9). OR., (ITEMP, LT, 0)) GOTO 6007
IIF(IPRT GE 30) WRITE(6, 216) ITEM
FORMAT(* FOUND NUMERIC DIGIT*, I2)
IFN=IFN*10+ITEMP
GOTO 1
10 SIGN=-1
IIF(IPRT GE 30) WRITE(6, 214)
FORMAT(* FOUND MINUS SIGN*)
1 CONTINUE
217 FORMAT(* SUBROUTINE IVAL$ RETURNING VALUE 'I20*)
IIF(IPRT GE 30) WRITE(6, 2225) IFN, SIGN
2225 FORMAT(* SET FINAL SIGN OF 'I20, ' WITH 'I20)

IVA00010
IVA00020
IVA00030
IVA00040
IVA00050
IVA00060
IVA00070
IVA00080
IVA00090
IVA0100
IVA0110
IVA0120
IVA0130
IVA0140
IVA0150
IVA0160
IVA0170
IVA0180
IVA0190
IVA0200
IVA0210
IVA0220
IVA0230
IVA0240
IVA0250
IVA0260
IVA0270
IVA0280
IVA0290
IVA0300
IVA0310
IVA0320
IVA0330
IVA0340
IVA0350
IVA0360
IVA0370
IVA0380
IVA0390
IVA0400
IVA0410
IVA0420
IVA0430
IVA0440
IVA0450
IVA0460
IVA0470
IVA0480
660 IVAL$=SIGN*IFN
   VAL=IVAL$
   IF(IPRT.GE.20)WRITE(6,217) VAL
   RETURN
C
C ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(* *** STRING LENGTH ERROR *** ',A4)
   IVAL$=-1
   STOP
6007 WRITE(6,6008)(A$(I),I=1,LA)
6008 FORMAT(* *** ATTEMPT TO FIND REAL VALUE OF ALPHABETIC STRING:*/
   2   ',110A1)
   STOP
   END
INTEGER FUNCTION LBL$(A$,LA, M, M1)

**LBLO0010**

C** STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN LBL INSTRUCTION. **LBLO0020**
C** THE RETURN VALUE OF THE FUNCTION LBL$ IS SET AS FOLLOWS: **LBLO0030**
C** 0 = CONTINUE TO COMPIL **LBLO0040**
C** -1 = AN ERROR IN COMPIL **LBLO0050**
C**ING THE INSTRUCTION. **LBLO0060**
C** **LBLO0070**
C** **LBLO0080

**LBLO0090**

**LBLO0100**

**LBLO0110**

**LBLO0120**

**LBLO0130**

**LBLO0140**

**LBLO0150**

**LBLO0160**

**LBLO0170**

**LBLO0180**

**LBLO0190**

**LBLO0200**

**LBLO0210**

**LBLO0220**

**LBLO0230**

**LBLO0240**

**LBLO0250**

**LBLO0260**

**LBLO0270**

**LBLO0280**

**LBLO0290**

**LBLO0300**

**LBLO0310**

**LBLO0320**

**LBLO0330**

**LBLO0340**

**LBLO0350**

**LBLO0360**

**LBLO0370**

**LBLO0380**

**LBLO0390**

**LBLO0400**

**LBLO0410**

**LBLO0420**

**LBLO0430**

**LBLO0440**

**LBLO0450**

**LBLO0460**

**LBLO0470**

**LBLO0480**

CONTAIN AN LBL$ INSTRUCTION.

A$ HAS BEEN IDENTIFIED AS FOLLOWS:

THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.

FUNCTION LBL$ IS SET AS FOLLOWS:

CONTINUE TO COMPILING THE INSTRUCTION.

AN ERROR IN COMPILING THE INSTRUCTION.
IF(LA-2) 25,50,6015

LENGTH OPERAND IMPIESES SINGLE CHARACTER ALPHA LOCAL LABEL

DO 30 I=1,26

INDEX=I

IF(A$(I).EQ. LABEL(I)) GOTO 35

CONTINUE

WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND

GOTO 6015

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

INDEX=INDEX+101

PROCESS A TWO BYTE INSTRUCTION

IBYTE=2
M(M1)=IBYTE

IF(IPRT.GE.20) WRITE(6,210)M1,IBYTE

FORMAT(' LBL$',15, 'LENGTH OF NEXT INSTR IS',13)

M1=M1+1

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

M(M1)=207

IF(IPRT.GE.10) WRITE(6,211)M1,M(M1)

FORMAT(' LBL$',15, 'TWO BYTE LBL INSTR',T75,13)

M1=M1+1

M(M1)=INDEX

IF(IPRT.GE.10) WRITE(6,212)M1,M(M1)

FORMAT(' LBL$',15, 'LBL 2D OPERAND ',T75,13)

M1=M1+1

LBL$=0

RETURN
LENGTH OF OPERAND IMPLIES MUST BE A NUMERIC LOCAL LABEL

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

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

IF(INDEX.GT.14) GOTO 40

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

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

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

M(M1)=1+INDEX
IF(IPRT.GE.10) WRITE(6,213) M1,M(M1)
FORMAT('"LBL$",I5,' ONE BYTE LBL INSTR',T75,I3)
M1=M1+1
LBL$=0
RETURN

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE

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

LOOK FOR ANOTHER QUOTE

PL=P2+1
P4=POS$(A$,LA,QUOTE,1,PL)
IF(P4) 6015,95,90
FOUND ANOTHER (THIRD OR MORE) QUOTE

P2=P4
GOTO 85

CHECK AFTER LAST QUOTE FOR KEY ASSIGNMENT CODE

LEFT=LA-P2
IF(LEFT) 6015,100,105

LA=LA-1
GOTO 120

PSTART=P2+1
CALL SEG$(A$,LA,SS1$,LSS1$,PSTART,LEFT)
K=IVAL$(SS1$,LSS1$,NO)
LA=LA-LEFT-1

SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL

LENGTH=LA-1
I BYTE=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

M(M1)=192
IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
FORMAT(" LBL",,I5," 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
POINTERs WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN. THIS
CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
BOUNDARIES WITHIN THE HP41CV MEMORY. THE CHAIN OF LABELS IS
RECOMPILED BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE
CHAIN ARE SET TO ZERO BY THIS COMPILER.

M(M1)=0
IF(IPRT.GE.10) WRITE(6,215) M1,M(M1)
FORMAT('LBL$',I5,'TRAILNG NULL INSTR',T75,I3)
M1=M1+1

SET INDICATOR FOR NUMBER OF ALPHA CHAR'S IN LABEL

M(M1)=241+LENGTH
IF(IPRT.GE.10) WRITE(6,216) M1,M(M1)
FORMAT('LBL$',I5,'LENGTH CODE ALPH LBL',',T75,I3)
M1=M1+1

ENCODE KEY ASSIGNMENT

IF(K.NE.0) GOTO 130
SINCE K=0 IMPLIES NULL KEY ASSIGNMENT
M(M1)=0
IF(IPRT.GE.10) WRITE(6,217) M1,M(M1)
FORMAT('LBL$',I5,'NULL KEY ASSIGNMENT',',T75,I3)
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=1ABS(K)
A1=K/10
B1=MOD(K,10)
K=16*(B1-1)+A1+K1
M(M1)=K
IF(IPRT.GE.10) WRITE(6,218) M1,M(M1)
FORMAT('LBL$',I5,'LBL KEY ASSIGNMENT',',T75,I3)
M1=M1+1

ADD ALPHABETIC CHARACTERS AND RETURN

LA=LENGTH
LBL$=ALPH$(A$,LA,M,M1)
RETURN
C
C ERROR HANDLING SECTIONIZES
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(1*** STRING LENGTH ERROR *** ',A4)
       WRITE(6,6010) LA, LB, IDIM
6010 FORMAT(1LA=',I10,' LB=',I10,' IDIM=',I10)
       LBL$=-1
       RETURN
6015 WRITE(6,6016)
6016 FFORMAT(1**** INVALID SECOND OPERAND IN LBL INSTR ****)
       LBL$=-1
       RETURN
       END
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-Z) **
COMMON/TEXT/IDIM, IPRT
COMMON/FLAGS/DONE,PODIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
LOGICAL DONE,PODIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTEGER*2 A$(IDIM)
C
INTEGER*2 BLNK/
INTEGER*2 IIND(3)/'I','N','D'/
INTEGER*2 C$(5)/'I','Z','Y','X','L'/
INTEGER*4 LC/5/
INTEGER*4 WHICH(5)
INTEGER*4 FUNC$/"MEM"/
INTEGER*4 M(1)
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT(1 TRACE ",13,1 MEM$:1',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C

STRIP STRING OF "MEM" CHARACTERS.

IF(LCUT$(A$,LA,3)) 6015,6015,7
IF(TRIM$(A$,LA1)) 6015,6015,10

ESTABLISH MOST LIKELY INSTR LENGTH AND PREFIX
I0
BYTE=2
PREFIX=WHICH(1)
H=WHICH(2)

CHECK FOR INDIRECT ADDRESS

**
C 20 P6=POSS$ (A$,LA,IND,3,1) ME M00490
   IF(P6) 6000,40,21 ME M00500
   INDIR=.TRUE. ME M00510
   IF (LCUT$(A$,LA,3)) 6000,6015,22 ME M00520
   IF (TRIMS$(A$,LA)) 6015,6015,40 ME M00530
C C C CHECK FOR NUMERIC SECOND OPERAND
C 40 IF (NUMC$ (A$,LA,IANSW)) 6015,45,60 ME M00540
C C C PROCESS NON-NUMERIC SECOND OPERAND
C 45 IF (LA.GT.1) GOTO 6015 ME M00550
   IZ=FIND$ (A$(1),LA,C$,LC,LCC) ME M00560
   IF (IZ.NE.0) GOTO 46 ME M00570
   WRITE(6,207) A$(1) ME M00580
   FORMAT (1,**** INVALID CHARACTER ***** ,5X,A1) ME M00590
   MEM$=-1 ME M00600
   RETURN ME M00610
   POSTFX=IZ+111 ME M00620
   GOTO 100 ME M00630
C C C PROCESS NUMERIC POSTFIX OPERANDS
C 60 POSTFX=IVAL$ (A$,LA,VAL) ME M00640
C C C CHECK FOR ONE BYTE VERSUS TWO BYTE NUMERIC OPERANDS
C 70 IF ( (POSTFX.GT.15).CR.(INDIR) ) GOTO 75 ME M00650
C C C PROCESS ONE BYTE NUMERIC SECOND OPERANDS
C   IBYTE=1 ME M00660
   PREFIX=POSTFX+H ME M00670
   GOTO 100 ME M00680
PROCESS TWO BYTE NUMERIC OPERANDS

CONTINUE

SET THE LENGTH OF THE INSTRUCTION

M(1) = IBYTE
IF(IPRT.GE.20) WRITE(6,215) M1,IBYTE
215 FORMAT(' MEM$','INSTR',LEN(M(1)))

ENCODE THE PREFIX OF THIS INSTRUCTION

M(1) = PREFIX
IF(IPRT.GE.10) WRITE(6,211) M1,(WHICH(I),I=3,5),M(M1)
211 FORMAT(' MEM$','INSTR',T7513)

ENCODE THE POSTFIX OF THIS INSTRUCTION

IF( IBYTE.GE.2) GOTO 125
IF(INDIR)POSTFX = POSTFX + 128
M(1) = POSTFX
IF(IPRT.GE.10) WRITE(6,221) M1,(WHICH(I),I=3,5),M(M1)
221 FORMAT(' MEM$','INSTR',T7513)

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
6001 FORMAT( '*** STRING LENGTH ERROR *** ' ,A4)
WRITE(6,6010) LA, LB, IDIM
6010 FORMAT( 'LA=','LB=','IDIM=')
MEM$=-1
RETURN
6015 WRITE(6,6016)
6016 FCMAT('***** INVALID SECOND OPERAND IN MEM INSTR ****')
MEM$=-1
RETURN
END
INTEGER FUNCTION NEWPG$(LINCNT, NUMPGE, TITLE$, LTITLE, MTITLE)

**NEW00010**

**NEW00020**

**NEW00030**

**NEW00040**

**NEW00050**

**NEW00060**

**NEW00070**

**NEW00080**

**NEW00090**

**NEW00100**

**NEW00110**

**NEW00120**

**NEW00130**

**NEW00140**

**NEW00150**

IMPLICIT INTEGER(A-Z)

COMMON/TEXT/IDIM/IPRT

INTEGER*2 TITLES$(LTITLE, MTITLE)

INTEGER*2 BLNK/

INTEGER*2 CARD$(80)

INTEGER*4 FUNC$(NEWPG$)

**NEW00160**

**NEW00170**

**NEW00180**

**NEW00190**

**NEW00200**

**NEW00210**

**NEW00220**

**NEW00230**

**NEW00240**

**NEW00250**

**NEW00260**

**NEW00270**

**NEW00280**

**NEW00290**

**NEW00300**

**NEW00310**

PRINT THE OUTPUT PAGE HEADING

**NEW00320**

**NEW00330**

**NEW00340**

**NEW00350**

**NEW00360**

**NEW00370**

**NEW00380**

**NEW00390**

**NEW00400**

**NEW00410**

**NEW00420**

**NEW00430**

**NEW00440**

**NEW00450**

**NEW00460**

**NEW00470**

**NEW00480**

WRITE(6,299)

IF(MTITLE.LE.0) GOTO 75

WRITE(6,200)(TITLE$(JJ,1), JJ=1, LTITLE)

IF(MTITLE.EQ.1) GOTO 75

DC 50 II = 2, MTITLE

WRITE(6,201)(TITLE$(JJ, II), JJ=1, LTITLE)

50 CONTINUE

UPDATE THE PAGE COUNTER AND RESET THE LINE COUNTER

NOMPGE = NUMPGE + 1

LINCNT = MTITLE

WRITE(6,297)
C   EXIT
    NEWPG$=NUMPGE
    RETURN

C   ERROR HANDLING SECTION FOLLOWS

C  6000 WRITE(6,6001) FUNC$
  6001 FORMAT(1X, '*** PAGE OUTPUT ERROR *** ', A4)
    NEWPG$=-1
    RETURN
    END
INTEGER FUNCTION NUMC$(A$,LA,IANSW)

C** THIS FUNCTION CHECKS TO SEE IF A STRING IS ALL NUMERIC.
C** NUMERIC IS DEFINED TO MEAN ALL DIGITS, "+", "-", ",", ",E", OR BLNK
C** THE RETURNED VALUE OF THE INSTRUCTION IS 0 IF NOT NUMERIC
C** 1 IF NUMERIC
C** -1 IF ERROR ENCOUNTERED

C** IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM)
INTEGER*4 FUNC$/'NUMC$/'
INTEGER*2 DIGIT(15)/"0","1","2","3","4","5","6","7","8","9",
2 IF(IPRT.GE.10) WRITE(6,200)LA,$A$(I),I=1,LA)
200 FORMAT(TRACE,'13,'NUMC$: '110A1')
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C DC 10 I=1,LA
C IF(LA.NE.1) GOTO 3
C NOT ALLOWED TO EXAMINE LAST 5 CHAR S IF LA IS ONE.
LOOK=10
GOTC 4
3 LOOK=15
DO 5 J=1,LOOK
IF($A$(I).EQ.DIGIT(J)) GOTO 10
5 CONTINUE
C 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
C CHARACTER FOUND TO BE NUMERIC, TAKE NEXT CHARACTER
CONTINUE
C IF YOU GET TO HERE THE STRING MUST BE ALL NUMERIC.
IANSW=1
NUMC$=1
IF(IPRT.GE.20) WRITE(6,202)
202 FORMAT(' STRING FOUND TO BE NUMERIC')
RETURN
C
C ERROR HANDLING SECTION follows
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ','A4)
WRITE(6,6010) LA,LB,IDIM
6010 FORMAT(' LA=',I10,',' ,LB=',I10,',' ,IDIM=',I10)
NUMC$=-1
RETURN
END
INTEGER FUNCTION PARS$(A$,LA$,B$,LB$)
C***********************************************************************************
C** STRING A$ IS SEARCHED FOR THE OCCURRENCE OF THE 1ST NON-LEADING THEIR IS SPLIT INTO TWO SUBSTRINGS, THE LEADING TOKEN (FIRST WORD) IS PLACED IN B$ AND THE REMAINDER IS PLACED IN A$.
C** AN ATTEMPT TO PARSE A NULL STRING WILL RESULT IN A SPECIAL CHARACTER BEING PLACED IN THE 1ST POSITION OF A$, A SUBSEQUENT ATTEMPT TO PARSE THIS STRING WILL RESULT IN A FATAL ERROR. THIS FEATURE IS INTENDED TO PREVENT UNCONTROLLED LOCPIING OF 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.
C***********************************************************************************
CIMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM),B$(IDIM)
INTEGER*2 BLNK/*'
INTEGER*2 HALT/*'
INTEGER*4 FUNC$/*PARS$'/
IF(IPRT.GE.10) WRITE(6,200)LA,$(A$(I),I=1,LA$)
200 FORMAT('TRACE ',I3,' PARS$ : ','A10A1')
IF(LA$.GT.IDIM.0R.LA$.LT.0) GOTO 6000
C
C
C CHECK TO SEE IF INPUT STRING IS NULL
C
C IF(LA$.N.E.0) GOTO 5
C IF(IPRT.GE.20) WRITE(6,204)
204 FORMAT(' ATTEMPTED TO PARSE A NULL STRING.')
IF(A$(I).EQ.HALT) GOTO 6005
A$(I)$=HALT
PARS$=0
LB=0
RETURN
C
C TRIM THE INPUT STRING OF LEADING BLANKS
C
LOCATE THE FIRST NON-LEADING BLANK IN A$ (THEREBY DETERMINE LB)

25    LB=0
      DO 30 I=1,LA
      IF(A$(I),.EQ.BLNK) GOTO 35
      LB=LB+1
      CONTINUE
      CONSTRUCT TOKEN

35    DO 45 I=1,LB
      B$(I)=A$(I)
      CONTINUE
      IF(IPRT.GE.20) WRITE(6,205) LB,(B$(I),I=1,LIB)
      FORMAT(' ',PARS$ FOUND TOKEN',I3,','""','0A1)
      REMOVE TOKEN FROM FRONT OF INPUT STRING

C    LEFT=LA-LB-1
      IF(LEFT.GT.0) GOTO 50
      LA=0
      GOTO 75

50    DO 95 I=1,LEFT
A$(I)=A$(I+LB+1)
CONTINUE
LA=LEFT

CHECK $A FOR TRAILING BLANKS
IF(LA.EQ.0) GOTO 75
IM=0
DO 60 I=1,LA
INDEX=LA-I+1
IF(A$(INDEX).NE.BLNK) GOTO 65
IM=IM+1
CONTINUE
60 CONTINUE
IF(IM.EQ.0) GOTO 75
IF(IPRT.GE.20) WRITE(6,207) IM
207 FORMAT(' FOUND',I3,' TRAILING BLANKS IN INPUT STRING')
IF(IM.LT.LA) GOTO 70
LA=0
PARS$=0
IF(IPRT.GE.20) WRITE(6,208)
208 FORMAT(' FOUND REMAINING STRING IS ALL BLANKS')
RETURN
70 LA=LA-IM
75 PARS$=LA
IF(IPRT.GE.20.AND.LA.EQ.0) WRITE(6,209)
209 FORMAT(' REMAINING STRING AFTER PARSE FUNCTION IS NULL')
RETURN

ERROR HANDLING SECTION FOLLOWS
6000 WRITE(6,6001) FUCNS
6001 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
PARS$=-1
RETURN
6005 WRITE(6,6006)
6006 FORMAT(' *** FATAL ERROR: ATTEMPTED TO PARSE A NULL STRING TWICE')
STOP
END
INTEGER FUNCTION POS$(A$, LA$, 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 FOUND. A MATCH IS ASSIGNED TO POS$. THE
SEARCH BEGINS AT LSTART$. IN A$, A ZERO IS RETURNED IF NO MATCH IS FOUND. A NEGATIVE NUMBER IS RETURNED UPON AN ERROR.

IMPLICIT INTEGER(A-Z)
COMMON/TEXT,IDIM,IPRT
INTEGER*2 A$(IDIM),B$(IDIM)
INTEGER*4 FUNC$/'POS$'/
IF(IPRT.GE.10) WRITE(6,200)LA$,A$(I),I=1,LA$
200 FORMAT('TRACE ',I3,' POS$: ',I10A1)
IF(LA$.GT.IDIM.OR.LA$.LT.0) GOTO 6000
IF(LB$.GT.IDIM.OR.LB$.LT.0) GOTO 6000

IF(LB$.LE.(LA$.LSTART+1))GOTO 10
POSS=0
IF(IPRT.GE.20) WRITE(6,201)
201 FORMAT('FIRST STRING TOO SHORT TO CONTAIN SECOND STRING')
RETURN

LEFT=LA$.LB$+1
DO 25 I=LSTART$,LEFT
DO 20 J=1,LB
JN=I+J-1
IF(IPRT.GE.20) WRITE(6,202)JN,A$(JN),J,B$(J)
202 FORMAT('COMPARE A$(',I3,')'=','A1,' WITH B$(',I3,')'=',A1)
IF(A$(JN).NE.B$(J)) GOTO 25
CONTINUE
POSS=I
IF(IPRT.GE.20) WRITE(6,203) I
203 FORMAT('AT POSITION ',I3,' SECOND STRING FOUND IN FIRST')
RETURN

CONTINUE
ROUTINE WILL FALL THROUGH TO FOLLOWING IF NO MATCH FOUND.
POSS=0
IF(IPRT.GE.20) WRITE(6,204)
204 FORMAT(' FIRST STRING SEARCHED AND SECOND STRING NOT FOUND')
RETURN
C ERROR HANDLING SECTION folLOWS
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(' *** STRING LENGTH ERROR *** ','A4)
POS$=-1
RETURN
END
INTEGER FUNCTION RCUT$(A$, LA, NUM)

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/IDIM, IPRT
INTEGER*2 A$(IDIM)
INTEGER*4 FUNC$/'RCUT$/
INTEGER*4 NUM
IF(IPRT,GE,10) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT('TRACE ',I3,' RCUT$ : ',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C LEFT=LA-NUM
20 IF((LEFT).GT.0) GOTO 20
202 IF(IPRT,GE,10) WRITE(6,202)
202 FORMAT(' STRING REDUCED TO NULL STRING BY RCUT$')
LA=0
RCUT$=0
RETURN
C
C
C CONTINUE
20 LA=LEFT
201 IF(IPRT,GE,20) WRITE(6,201)LA,(A$(I),I=1,LA)
201 FORMAT(' STRING NOW ',I4,' ',110A1)
RCUT$=LA
RETURN
C
C ERROR HANDLING SECTION FOLLOWS
C
6000 WRITE(6,6001) FUNC$
601 FORMAT(' *** STRING LENGTH ERROR *** ',A4)
RCUT$=-1
RETURN
END
INTEGER FUNCTION SEG$(A$,LA,B$,LB,LSTART,LOUT)

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

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
INTEGER*2 A$(IDIM),B$(IDIM)
INTEGER*2 BLNK,'/','/
INTEGER*4 FUNC$/'SEG'/
LOGICAL FIRST
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)

200 FORMAT('TRACE ',I3,' SEG$ : ',110A1)
FIRST=.TRUE.
IF(LSTART.GT.IDIM.OR.LSTART.LT.0) GOTO 6000
IF(LOUT.GT.IDIM.OR.LOUT.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(6,204) LSTART,LOUT

204 FORMAT(' START POINT ('',I3,'') GREATER THAN LENGTH FIRST STRING')

2 /* NEW STRING HAS ',I3,', BLANKS' */
SEG$=0
RETURN

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

ERROR HANDLING SECTION FOLLOWS

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

**THIS FUNCTION STRIPS A STRING OF LEADING AND TRAILING BLANKS.**

**THE RETURN VALUE OF THE FUNCTION IS SET TO THE LENGTH OF THE STRING REMAINING AFTER THE BLANKS ARE REMOVED.**

**COMMON/TEXT/IDIM, JPR1M**

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

**INTEGER*2 BLNK**

**INTEGER*4 FUNC$/'TRIM'/**

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

200 FORMAT(' TRACED ',I3,' TRIM$: ',I10A1)

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

58  IM=J

DC 60 I=1,LA
INDEX=LA-I+1
IF(A$(INDEX).NE.BLNK) GOTO 65

65  IM=IM+1

CONTINUE

IF(IM.EQ.0) GOTO 70

65  IF(IPRT.GE.20) WRITE(6,207) IM

207  FORMAT(' FOUND ',I3,' TRAILING BLANKS IN STRING')

IF(IM.NE.LA) GOTO 70

LA=0
TRIM$=0
IF(IPRT.GE.20) WRITE(6,208)

208  FORMAT('FOUND STRING IS ALL BLANKS')

RETURN

70  IEND=LA-IM

CHECK A$ FOR TRAILING BLANKS

DO 10 I=1, IEND

10  IF(A$(I).NE.BLNK) GOTO 15

CHECK A$ FOR LEADING BLANKS

DO 10 I=1, IEND

10  IF(A$(I).NE.BLNK) GOTO 15
IM=IM+1
10 CONTINUE
15 IF(IM.EQ.0) GOTO 25
   IF(IPRT.GE.20) WRITE(6,211) IM
   FORMAT(‘FOUND’,I3,’ LEADING BLANKS IN STRING’)
211 IBEG=1+IM

C C C
C DETERMINE LENGTH OF INPUT STRING
C LA=IENS-IBEG+1
IF(LA.LE.IDIM) GOTO 30
   LOSS=LA-IDIM
   IF(IPRT.GE.10) WRITE(6,216) LOSS
   FORMAT(‘ STRING TOO LONG FOR MAX STRING LENGTH. LOST’,I3)
216 LA=IDIM
   IENS=IENS-LOSS
C C C
C TRANSFER THE A$ CHARACTERS TO THE INPUT STRING.
C INDEX=1
30 DO 85 I=IBEG, IENS
   A$(INDEX)=A$(I)
   IF(IPRT.GE.30) WRITE(6,209) I,A$(I),INDEX,A$(INDEX),
   FORMAT(‘ MOVE A$(‘,I3,’)=‘,A1,’ IS NOW A$(‘,I3,’)=‘,A1)
   INDEX=INDEX+1
85 CONTINUE

C C C
C CHECK FOR STRING ERROR AND RETURN
C IF(IPRT.GE.10) WRITE(6,222)LA, (A$(I), I=1,LA)
   FORMAT(‘ TRIM$ ‘,I3,’ AFTER : ‘,11)A1)
222 IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

C C
C TRIM$=LA
C RETURN

C C ERROR HANDLING SECTION Follows
C
6000 WRITE(6,6001) FUNC$
6001 FORMAT(‘*** STRING LENGTH ERROR *** ’,A4)
   WRITE(6,6010) LA, LB, IDIM
6010  FCRMAT(' LA='IL0,' LB='IL0,' IDIM='IL0)
TRIM$=-1
RETURN
END

TRI00970
TRI00980
TRI00990
TRI01000
INTEGER FUNCTION XEQ$(A$, LA, M, M1)  

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

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

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

**Implicit INTEGER(A-Z)**  

COMMON/TEXT/IDIM, IPRT  
COMMON/FLAGS/DONE, POQ, DIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2  
COMMON/CNTR/P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2  
LOGICAL DONE, POQ, DIGIT, PALPHA, DIGIT, ALPHA, INDIR, FLAG1, FLAG2  
INTEGER*4 P1, P2, P3, P4, P5, P6, P7, P8, P9, S1, S2  
INTEGER*2 A$(IDIM),  
INTEGER*2 BLNK/*  
INTEGER*2 Q*/*  
INTEGER*2 IND(3)/'I','N','D'/  
INTEGER*2 LABEL(26)/'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'/  
2  
INTEGER*4 FUNC$*/'XEQ*/  
INTEGER*4 M(1)  
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I), I=1,LA)  

200  
FORMAT('TRACE ',I3, ' XEQ$ : ',110A1)  
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000  

**Establish Default Prefix and Instruction Length Values**  
(THESE ARE THE VALUES FOR 3 BYTE LOCAL NUMERIC XEQ WITHOUT IND)  

IBYTE=3  
PREFI=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 LCUT$(A$,LA,P1)
IF(IPRT.GE.20)WRITE(6,235)
FORMAT('DETECTED INDIRECT XEQ INSTRUCTION')
INDIR=TRUE.
IBYTE=2
PREFIX=174

CHECK FOR NUMERIC OPERAND

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

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

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

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

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

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

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

HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPERAND.
AND THEN GOTO PROCESS A THREE BYTE INSTRUCTION.
INDEX = INDEX + 101
GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

IF(I$VAL$(A$,,LA,,INDEX))=6015,,75,,75

PROCESS THE GOTO INSTRUCTION (OPAND>14)

M(M1) = I$BYTE
IF(IPRT,GE,20) WRITE(6,210) M1,, I$BYTE
210 FORMAT('', XEQ$,',',I5,,,' LENGTH OF NEXT INSTR IS',',I3)
M1 = M1 + 1

LOAD THE XEQ INSTR PREFIX

M(M1) = PREFIX
IF(IPRT,GE,10) WRITE(6,211) M1,, M(M1)
211 FORMAT('', XEQ$,',',I5,,,' XEQ PREFIX',',INSTR',',T75,,I3)
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)
221 FORMAT('', XEQ$,',',I5,,,' NULL FOR XEQ INSTR',',T75,,I3)
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(' XEQ$','I5',' XEQ 2D OPERAND ', 'T75','I3)       XEQQ1450
       XEQQ1460
       XEQQ1470
       XEQQ1480
       XEQQ1490
       XEQQ1500
       XEQQ1510
       XEQQ1520
       XEQQ1530
       XEQQ1540
       XEQQ1550
       XEQQ1560
       XEQQ1570
       XEQQ1580
       XEQQ1590
       XEQQ1600
       XEQQ1610
       XEQQ1620
       XEQQ1630
       XEQQ1640
       XEQQ1650
       XEQQ1660
       XEQQ1670
       XEQQ1680
       XEQQ1690
       XEQQ1700
       XEQQ1710
       XEQQ1720
       XEQQ1730
       XEQQ1740
       XEQQ1750
       XEQQ1760
       XEQQ1770
       XEQQ1780
       XEQQ1790
       XEQQ1800
       XEQQ1810
       XEQQ1820
       XEQQ1830
       XEQQ1840
       XEQQ1850
       XEQQ1860
       XEQQ1870
       XEQQ1880
       XEQQ1890
       XEQQ1900
       XEQQ1910
       XEQQ1920

PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE

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

CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS

85    LEFT=LA-P2
       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

120   LENGTH=LA-1
       FOR GTO H=2  FOR LBL H=4  FOR XEQ H=2
       H=2
       LBYTE=H+LENGTH
       M(M1)=LBYTE
       IF(IPRT.GE.20)WRITE(6,214)M1,LBYTE
       M1=M1+1

HAVE FOUND VALID ALPHANUMERIC LABEL, LCAD "XEQ" INSTRUCTION

226

PREFIX=30
       FOR GTO PREFIX=29  FOR LBL PREFIX=192  FOR XEQ PREFIX=30
       M(M1)=PREFIX
214 IF(IPRT.GE.10) WRITE(6,214) M1, M(M1)
FORMAT(' XEQ$,I5,' ALPH$ 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$,I5,' LENGTH CODE ALPH$ XEQ$,T75,I3)
M1=M1+1

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

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

IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM, IPRT
INTEGER*2 A$(IDIM)
INTEGER*2 COMA/*', ' /, S$1$(40)
INTEGER*4 FUNC/*XRO1*/
INTEGER*4 M(1)
REAL*4 RFrac, RDIFF

IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200 FORMAT(* TRACEx13,* XRO$: '110A1')
IF(LA.LT.IDIM.OR.LA.LT.0) GOTO 6000

C

7 IF(LCUT$(A$, LA, 4)) 6015, 6020, 7
10 IF(TRIM$(A$, LA)) 6015, 6020, 10
12 IF(SEG$(A$, LA, S$1$, L$SS1$, 1, P1)) 6015, 6020, 12
15 IF(IVAL$(S$1$, L$SS1$, ROM)) 6030, 20, 20
20 IF(IVAL$(A$, LA, PGM)) 6030, 25, 25
25 IF(RFrac=ROM)/4
1FIRST=160+IFrac
RFrac=FLOAT(ROM)/4.0
RDIFF=RFrac-FLOAT(IFrac)
RDIFF=256.0*RDIFF
IDIF=INT(RDIFF)
ISECND=PGM+IDIF
SET THE LENGTH OF THE INSTRUCTION

800   BYTE=1
   M(1)=BYTE
   IF(IPRT.GE.20)WRITE(6,215)M1,1BYTE
   FORMA1' XRO$,15,' LENGTH OF THIS INSTR IS',13
   M1=M1+1

ENCOD THE PREFIX OF THIS INSTRUCTION

211   M(1)=IFIRST
   IF(IPRT.GE.10)WRITE(6,211)M1,IFIRST,ROM,M(1)
   FORMA1' XRO$,15,' PREFIX=',13,' ROM=',13,T75,13
   M1=M1+1

ENCOD THE POSTFIX OF THIS INSTRUCTION

321   M(1)=ISECOND
   IF(IPRT.GE.10)WRITE(6,221)M1,ISECOND,PGM,M(1)
   FORMA1' XRO$,15,' POSTFIX=',13,' PGM=',13,T75,13
   M1=M1+1

125   XRO$=0
   RETURN

ERROR HANDLING SECTION FOLLOWS

6000  WRITE(6,6001) FUNC$
6001  FORMA1' *** STRING LENGTH ERROR *** ',A4)
6010  WRITE(6,6010) LA, IDIM
       FORMA1' LA=',110,' IDIM=',110)
       XRO$=-1
       RETURN
6015  WRITE(6,6016)
6016  FCMAT1' **** INVALID ROM NUMBER IN XRO INSTR ****')
       XRO$=-1
       RETURN
6020  WRITE(6,6021)
6021 FORMAT('****** INVALID PROGRAM NUMBER IN XRO INSTR *****')
XRO$=-1
RETURN
6030 WRITE(6,6031)
6031 FORMAT('****** NUMERIC CONVERSION ERROR IN XRO INSTR *****')
XRO$=-1
RETURN
END
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 TWICE THE WIDTH OF A UNIT BAR -- --IT IS THE WIDTH
OF A ONE BAR.
DOUBLE=0.010*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.
The title of the program is the height of the title characters.

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

TSPACE=CSPACE*(1.5*DSPACE)

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

PLUS THE SPACE BETWEEN THE BARS, INCLUDING LABELS

HFACTR=1.7*HEIGHT

PLONG IS THE HEIGHT OF EACH PAGE OF BARCODE

PLONG=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

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

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

CALL PLOTS(1,0,0)

CALL PLOT(0.0,0.0,3)

IROW=0

IPRT=0

X=TMAR

Y=ZERO

READ THE TITLE OF THE BARCODE PROGRAM FROM THE INPUT FILE

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

FORMAT(80A1)

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

READ(5,101,END=30000)(IN(I),I=1,132)

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

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

FORMAT(,132A1)
WRITE THE TITLE ON THE PLOT

IF(MOD(IROW, PERPAGE) .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, YM1, 2)
CALL PLOT(XM1, YM2, 2)
CALL PLOT(XM2, YM2, 2)
CALL PLOT(XM1, YM1, 2)
CALL PLOT(X, Y, 3)
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

ROWNUM = ROWNUM + 1.0
IROW = IROW + 1
CALL NEWPEN(2)
CALL NUMBER(X, Y, CSIZE, ROWNUM, 90,), 1
CALL NEWPEN(NIBS)
X = X + CSPACE
CALL PLOT(X, Y, 3)

CONVERT THE ZERO'S AND ONE'S INTO BARS OF CORRECT WIDTH

DC 1000 I = 1, 132

CHECK FOR ZERO OR ONE BIT

IF(IN(I).EQ.ZER) GOTO 1)
IF(IN(I).EQ.ONE) GOTO 2)
IF(IN(I).EQ.BLNK) GOTO 2000
C DRAW A ZERO BAR OF UNIT WIDTH
100  X=X+HEIGHT
     CALL PLOT(X,Y,2)
     Y=Y+DOUBLE
     X=X-HEIGHT
     CALL PLOT(X,Y,3)
     GOTO 1000

C

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

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

C 3000  CALL PLOT(0,0,0,999)
         STOP
         END
The following is the data set read by the cross compiler.

<table>
<thead>
<tr>
<th>HP41C SOURCE</th>
<th>CCDE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>62</td>
</tr>
<tr>
<td>+</td>
<td>64</td>
</tr>
<tr>
<td>-</td>
<td>65</td>
</tr>
<tr>
<td>*</td>
<td>66</td>
</tr>
<tr>
<td>/</td>
<td>67</td>
</tr>
<tr>
<td>&lt;</td>
<td>68</td>
</tr>
<tr>
<td>&gt;</td>
<td>69</td>
</tr>
<tr>
<td>=</td>
<td>70</td>
</tr>
<tr>
<td>&amp;</td>
<td>71</td>
</tr>
<tr>
<td>1</td>
<td>72</td>
</tr>
<tr>
<td>HMS+</td>
<td>73</td>
</tr>
<tr>
<td>HMS-</td>
<td>74</td>
</tr>
<tr>
<td>MOD</td>
<td>75</td>
</tr>
<tr>
<td>+</td>
<td>76</td>
</tr>
<tr>
<td>-</td>
<td>77</td>
</tr>
<tr>
<td>P-R</td>
<td>78</td>
</tr>
<tr>
<td>R-P</td>
<td>79</td>
</tr>
<tr>
<td>LN</td>
<td>80</td>
</tr>
<tr>
<td>SQRT</td>
<td>81</td>
</tr>
<tr>
<td>Y</td>
<td>x</td>
</tr>
<tr>
<td>CHS</td>
<td>83</td>
</tr>
<tr>
<td>E</td>
<td>x</td>
</tr>
<tr>
<td>LOG</td>
<td>85</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
</tr>
<tr>
<td>E</td>
<td>x-1</td>
</tr>
<tr>
<td>SIN</td>
<td>88</td>
</tr>
<tr>
<td>COSH</td>
<td>89</td>
</tr>
<tr>
<td>TAN</td>
<td>90</td>
</tr>
<tr>
<td>ASIN</td>
<td>91</td>
</tr>
<tr>
<td>ACCS</td>
<td>92</td>
</tr>
<tr>
<td>ATAN</td>
<td>93</td>
</tr>
<tr>
<td>LEC</td>
<td>94</td>
</tr>
<tr>
<td>1/x</td>
<td>95</td>
</tr>
<tr>
<td>ABS</td>
<td>96</td>
</tr>
<tr>
<td>FACT</td>
<td>97</td>
</tr>
<tr>
<td>X#C?</td>
<td>98</td>
</tr>
<tr>
<td>X&gt;C?</td>
<td>99</td>
</tr>
<tr>
<td>LN+X</td>
<td>100</td>
</tr>
<tr>
<td>X&gt;C?</td>
<td>101</td>
</tr>
<tr>
<td>X=C?</td>
<td>102</td>
</tr>
<tr>
<td>INT</td>
<td>103</td>
</tr>
<tr>
<td>R-R</td>
<td>104</td>
</tr>
<tr>
<td>R-C</td>
<td>105</td>
</tr>
<tr>
<td>HMS</td>
<td>106</td>
</tr>
<tr>
<td>HR</td>
<td>107</td>
</tr>
<tr>
<td>RNG</td>
<td>108</td>
</tr>
<tr>
<td>UCT</td>
<td>109</td>
</tr>
<tr>
<td>CLE</td>
<td>110</td>
</tr>
<tr>
<td>X&lt;Y</td>
<td>111</td>
</tr>
<tr>
<td>PI</td>
<td>112</td>
</tr>
<tr>
<td>CLST</td>
<td>113</td>
</tr>
<tr>
<td>PI</td>
<td>114</td>
</tr>
<tr>
<td>RDA</td>
<td>115</td>
</tr>
<tr>
<td>LASTX</td>
<td>116</td>
</tr>
</tbody>
</table>

235.
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLX</td>
<td>115</td>
<td>31</td>
<td>29</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>X=1?</td>
<td>120</td>
<td>45</td>
<td>99</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>X#Y?</td>
<td>121</td>
<td>91</td>
<td>97</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SIGN</td>
<td>122</td>
<td>78</td>
<td>75</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>XC=CG?</td>
<td>123</td>
<td>93</td>
<td>91</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>WEND</td>
<td>124</td>
<td>45</td>
<td>57</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SD?</td>
<td>125</td>
<td>76</td>
<td>74</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>AVE</td>
<td>126</td>
<td>22</td>
<td>20</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>CLG</td>
<td>127</td>
<td>28</td>
<td>26</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>CEQ</td>
<td>128</td>
<td>35</td>
<td>33</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>RAD</td>
<td>129</td>
<td>69</td>
<td>68</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>GRAD</td>
<td>130</td>
<td>48</td>
<td>46</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>ENTER</td>
<td>131</td>
<td>38</td>
<td>37</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>STP</td>
<td>132</td>
<td>86</td>
<td>84</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>RTN</td>
<td>133</td>
<td>73</td>
<td>72</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>BEEP</td>
<td>134</td>
<td>23</td>
<td>21</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>CLA</td>
<td>135</td>
<td>27</td>
<td>25</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ASF</td>
<td>136</td>
<td>18</td>
<td>16</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>PSE</td>
<td>137</td>
<td>66</td>
<td>54</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>CLR</td>
<td>138</td>
<td>29</td>
<td>27</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>ARF</td>
<td>139</td>
<td>15</td>
<td>13</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>AC</td>
<td>140</td>
<td>16</td>
<td>14</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>GFF</td>
<td>141</td>
<td>62</td>
<td>60</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FRMPT</td>
<td>142</td>
<td>65</td>
<td>63</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>ADV</td>
<td>143</td>
<td>14</td>
<td>12</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>RCL</td>
<td>144</td>
<td>70</td>
<td>69</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>STC</td>
<td>145</td>
<td>65</td>
<td>63</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ST-</td>
<td>146</td>
<td>82</td>
<td>79</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ST+</td>
<td>147</td>
<td>45</td>
<td>41</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ST/</td>
<td>148</td>
<td>81</td>
<td>80</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ISG</td>
<td>149</td>
<td>54</td>
<td>52</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CSE</td>
<td>150</td>
<td>36</td>
<td>34</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>VIEW</td>
<td>151</td>
<td>89</td>
<td>87</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>GREG</td>
<td>152</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>ASY</td>
<td>153</td>
<td>20</td>
<td>18</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>ARCL</td>
<td>154</td>
<td>17</td>
<td>15</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>FIX</td>
<td>155</td>
<td>44</td>
<td>42</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>SCI</td>
<td>156</td>
<td>44</td>
<td>42</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ENG</td>
<td>157</td>
<td>75</td>
<td>73</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>TGNE</td>
<td>158</td>
<td>37</td>
<td>36</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>159</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>160</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>161</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>162</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>163</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>164</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>165</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>166</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>XRCM</td>
<td>167</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>SF</td>
<td>168</td>
<td>77</td>
<td>75</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>CCF</td>
<td>169</td>
<td>24</td>
<td>22</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FSTC</td>
<td>170</td>
<td>47</td>
<td>45</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>FC3C</td>
<td>171</td>
<td>43</td>
<td>41</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>FS2</td>
<td>172</td>
<td>46</td>
<td>44</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>FC2</td>
<td>173</td>
<td>42</td>
<td>40</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>X&lt; &gt;</td>
<td>206</td>
<td>52</td>
<td>88</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
LIST OF REFERENCES

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


### INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Initial Distribution List</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Technical Information Center  
Cameron Station  
Alexandria, Virginia 22314 |
| 2.  | 2      | Library, Code 0142  
Naval Postgraduate School  
Monterey, California 93940 |
| 3.  | 1      | Department Chairman, Code 55  
Department of Operations Research  
Naval Postgraduate School  
Monterey, California 93940 |
| 4.  | 5      | Associate Professor S. H. Parry, Code 55Py  
Department of Operations Research  
Naval Postgraduate School  
Monterey, California 93940 |
| 5.  | 5      | Associate Professor R. H. Shudde, Code 55Su  
Department of Operations Research  
Naval Postgraduate School  
Monterey, California 93940 |
| 6.  | 1      | Office of the Commanding General  
U. S. Readiness Command  
ATTN: General Donn A. Starry  
MacDill AFB, Florida 33621 |
| 7.  | 1      | Office of the Commanding General  
U.S. Army Training and Doctrine Command  
ATTN: General Glenn Otis  
Fort Monroe, Virginia 23651 |
| 8.  | 1      | Chief  
TRADOC Research Element Monterey  
Naval Postgraduate School  
Monterey, California 93940 |
| 9.  | 1      | Headquarters  
U. S. Army Training and Doctrine Command  
ATTN: Director, Studies and Analysis Directorate  
Mr. S. Goldberg  
Fort Monroe, Virginia 23651 |
| 10. | 1      | Headquarters  
U. S. Army Training and Doctrine Command  
ATTN: ATCG-T (BG Morelli)  
Fort Monroe, Virginia 23651 |
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: ATTN-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: DAMO-2D
Washington, D.C. 20310
<table>
<thead>
<tr>
<th>21. Commander</th>
</tr>
</thead>
<tbody>
<tr>
<td>U. S. Army Concepts Analysis Agency</td>
</tr>
<tr>
<td>ATTN: MOCA-WG (LTC Earl Darden)</td>
</tr>
<tr>
<td>8120 Woodmont Avenue</td>
</tr>
<tr>
<td>Bethesda, Maryland 20014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>22. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>U. S. Army Material Systems Analysis Activity</td>
</tr>
<tr>
<td>ATTN: DRXSY-CM (Mr. Bill Niemeyer)</td>
</tr>
<tr>
<td>Aberdeen Proving Grounds, Maryland 21005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>U. S. Army Night Vision and Electro-Optical Lab</td>
</tr>
<tr>
<td>ATTN: DEL-NV-01 (Mr. Frank Shields)</td>
</tr>
<tr>
<td>Fort Belvoir, Virginia 22060</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>24. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>U. S. Army TRADOC Systems Analysis Activity</td>
</tr>
<tr>
<td>ATTN: Mr. Ray Heath</td>
</tr>
<tr>
<td>White Sands Missile Range, New Mexico 88002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>25. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combat Developments Studies Division</td>
</tr>
<tr>
<td>ATTN: MAJ W. Scott Wallace</td>
</tr>
<tr>
<td>U. S. Army Armor Agency</td>
</tr>
<tr>
<td>Fort Knox, Kentucky 40121</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>26. Commandant</th>
</tr>
</thead>
<tbody>
<tr>
<td>U. S. Army Field Artillery School</td>
</tr>
<tr>
<td>ATTN: ATSF-MBT (CPT Steve Starner)</td>
</tr>
<tr>
<td>Fort Sill, Oklahoma 73503</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>27. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combat Developments</td>
</tr>
<tr>
<td>U. S. Army Aviation Agency</td>
</tr>
<tr>
<td>Fort Rucker, Alabama 36362</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>28. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combat Developments</td>
</tr>
<tr>
<td>U. S. Army Infantry School</td>
</tr>
<tr>
<td>Fort Benning, Georgia 31905</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>29. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armored Combat Vehicle Technology Program</td>
</tr>
<tr>
<td>ATTN: COL Fleming</td>
</tr>
<tr>
<td>U. S. Army Armor Center</td>
</tr>
<tr>
<td>Fort Knox, Kentucky 40121</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>30. Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combat Developments</td>
</tr>
<tr>
<td>ATTN: MAJ William D. Meiers</td>
</tr>
<tr>
<td>U. S. Army Air Defense Agency</td>
</tr>
<tr>
<td>Fort Bliss, Texas 79905</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31. Commander</th>
</tr>
</thead>
<tbody>
<tr>
<td>U. S. Army Logistics Center</td>
</tr>
<tr>
<td>ATTN: ATCL-OS (Mr. Cameron/CPT Schuessler)</td>
</tr>
<tr>
<td>Fort Lee, Virginia 23801</td>
</tr>
</tbody>
</table>
32. Director  
U. S. Army Material Systems Analysis Agency  
ATTN: DRXSY-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 N. E. Circle Blvd.  
Corvallis, Oregon 97330

42. Mr. Richard Nelson, Editor  
PPC Journal  
5541 W. Camden Place  
Santa Ana, California 92704
ATTN: DAP

Department of the Army
Office of the Deputy Chief of Staff for Personnel
ATTN: DAPE-MPE-SS (Captain James Richmann)
Washington, D.C. 20310